Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FIN DE GRADO

TÍTULO DEL TFG: Control de vuelo de un dron utilizando un acelerómetro

TITULACIÓN: Grado en Ingeniería en Aeronavegación

AUTOR: Mónica Milán Muñoz

DIRECTOR: Pablo Royo Chic y Oscar Casas Piedrafita

FECHA: 8 de Febrero de 2017

Título: Control de vuelo de un dron utilizando un acelerómetro

Autor: Mónica Milán Muñoz

Director: Pablo Royo Chic y Oscar Casas Piedrafita

Fecha: 8 de Febrero de 2017

Resumen

A día de hoy el uso de los vehículos aéreos no tripulados está empezando a ser cada vez más popular. Con la intención de poder integrarlos en espectáculos artísticos, este proyecto expone el diseño de un nuevo sistema de control de vuelo que facilite el pilotaje al usuario.

La forma de control definida se basa en los movimientos voluntarios de la mano, por lo que resulta mucho más intuitivo que cualquier otro medio convencional y además conlleva una gran maniobrabilidad. La arquitectura completa consta de dos partes claramente diferenciadas, por una parte el subsistema que detecta la actividad de la mano y por otra el dron.

La adquisición de movimientos se basa en valores de aceleración y se realiza mediante el reloj eZ430-Chronos de Texas Instruments, que dispone de un acelerómetro de tres ejes. A dichos datos se les aplica un preprocesado para determinar la dirección del movimiento, y más tarde un filtrado para ignorar temblores o pequeños movimientos involuntarios y así obtener un valor estable.

Se han estudiado y evaluado dos plataformas distintas para el subsistema del dron, el Crazyflie 2.0 de Bitcraze y el AR.Drone 2.0 de Parrot ambos cuadricópteros. Considerando la poca estabilidad del primero, se ha determinado que el AR.Drone 2.0 es más apropiado para la finalidad que persigue este proyecto.

En relación a la modelización, se han definido una serie de posiciones de equilibrio y movimientos de interés de la mano, para realizar el envío de comandos al cuadricóptero.

Finalmente, se ha integrado la adquisición de datos del reloj con el vehículo aéreo no tripulado. Tras múltiples ensayos y pruebas de vuelo, se ha concluido que la respuesta que se obtiene es adecuada para la aplicación propuesta, tanto en términos de estabilidad como en rapidez del sistema.

Title: Accelerometer based flight control system for drones

Author: Mónica Milán Muñoz

Director: Pablo Royo Chic and Oscar Casas Piedrafita

Date: February 8, 2017

Overview

Nowadays the use of unmanned aerial vehicles is becoming increasingly popular. With the intent of integrate them in artistic performances, such as dance, this project exposes the design a new flight control system, which makes piloting easier to the user.

The defined control system is based on voluntary hand movements, consequently it results to be much more intuitive than conventional means, as well as it allows a high manoeuvrability. Full architecture is composed by two clearly different parts, the subsystem that detects hand movements and then the drone.

Movement acquisition is based on acceleration values and it is carried out by using eZ430-Chronos watch by Texas Instruments, which has integrated a three-axis accelerometer. Afterwards, obtained data is pre-processed to determine movement direction, then a filter is applied in order to disregard hand trembling or minor involuntary hand movements, thus a stable value is achieved.

It has been studied and evaluated two different flying devices, the Crazyflie 2.0 by Bitcraze and the AR.Drone 2.0 by Parrot, both quadcopters. Considering the low stability of the first one, it has been determined that the AR.Drone 2.0 is more appropriate according to the purpose of this project.

It has been carried out a modelization that consists in the definition of some equilibrium positions as well as movements of interest, so as to send commands to the quadcopter.

Finally, it has been performed subsystems integration and after multiple flight tests, it has been concluded that the resulting system is adequate to the proposed application, not only in terms of stability but also in time response.

Quiero dar las gracias en primer lugar a Pablo, mi tutor, por la oportunidad que me dio sin tan siquiera conocerme, por guiarme y por toda su ayuda. Tampoco puedo dejar de agradecer a Raúl sus consejos e infinita paciencia, ni a Óscar, mi codirector, ni a todas las personas que me han ayudado de un modo u otro con valiosas observaciones. Gracias también a familia y amigos por el apoyo, por aguantarme y creer en mí. Especialmente a Sonia y a Ana por darme toneladas de energía.

Un grand merci également à : mes colocataires, Giulia (notre infiltrée préférée), Aurélie, Pauline, Sergio, Anshuman et mes collègues de l'IMA. Pour me faire réaliser que comme dit la chanson, c'est bien vrai que « le bonheur c'est pas le but mais le moyen et pas la chute mais le chemin ».

ÍNDICE

Lista de acrónimos

UAS Unmanned Aircraft System
ICARUS Intelligent Communications and Avionics for Robust UAS
API Application Programming Interface
IDE Integrated Development Environment
IMU Inertial Measurement Unit
LCD Liquid Crystal Display
PEP Python Enhancement Proposal
RF Radio Frequency
PC Personal Computer
GUI Graphical User Interface
SDK Software Development Kit
UDP User Datagram Protocol
TCP Transmission Control Protocol
CSV Comma Separated Values
ROS Robot Operating System

Lista de figuras

INTRODUCCIÓN

En la actualidad pese a que las leyes y regulaciones que los rigen no están completamente definidas [1], el uso de los vehículos aéreos no tripulados (UAS) se ha extendido mucho más allá de propósitos meramente militares. Industrias como la cinematográfica [2], la agrícola [3], o incluso empresas de reparto [4] apuestan por emplear drones para realizar ciertas tareas. Además, existe un creciente interés y el mercado está sufriendo una gran expansión debido a la enorme cantidad de fines lúdicos y distintas posibilidades que ofrecen [5].

Este proyecto, consiste en el diseño y verificación de un sistema de control de vuelo basado en un acelerómetro de tres ejes, que permite el pilotaje de un dron a partir del movimiento voluntario de la mano de una persona.

La memoria se divide fundamentalmente en cuatro capítulos. Primero se presentan las dos partes que conforman la arquitectura completa: el subsistema de adquisición de movimientos y el vehículo aéreo no tripulado, en el primer y segundo capítulo respectivamente. El tercer capítulo habla sobre la modelización y detección de los movimientos, mientras que el cuarto trata de la integración de sistemas y los resultados obtenidos. En el quinto capítulo se exponen las conclusiones y trabajo futuro, a continuación las referencias y finalmente se adjuntan cinco anexos con información adicional y características técnicas.

Objetivos

Los principales objetivos del proyecto se describen a continuación.

1. **Diseño de un sistema de adquisición de movimientos.**
   El sistema debe realizar la obtención de datos del acelerómetro, y posteriormente preprocesar y filtrar los valores adquiridos.

2. **Control del vuelo del dron.**
   El dron tiene que poder ser guiado de forma estable.

3. **Definición del modelo de control.**
   Se tienen que definir, caracterizar y cuantificar los movimientos a efectuar por el usuario para controlar el vuelo del dron. También se tienen que establecer posiciones estables o puntos de equilibrio.

4. **Detección de los movimientos.**
   A partir de los datos adquiridos del acelerómetro, se tiene que poder determinar si se está en una posición de equilibrio. En caso contrario, es decir, si hubiese algún tipo de movimiento, habría que especificar la dirección y sentido del mismo.

5. **Integración de sistemas.**
   Se tiene que conseguir coordinar en tiempo real la adquisición y detección de movimientos con el envío de órdenes al vehículo aéreo no tripulado.

Motivación

La principal motivación de este proyecto, surge de la idea de poder facilitar la integración de los drones en espectáculos, o ámbitos artísticos como la danza, simplificando el pilotaje, de forma que el control sea mucho más intuitivo y fácil de cara al usuario.

Además para el público general, en ocasiones no resulta del todo fácil pilotarlos. Por lo que el sistema propuesto, también se ha concebido para evitar la necesidad de joysticks o mandos tradicionales. De forma que permite el control del vuelo simplemente con el movimiento de la mano, dado que es una de las partes del cuerpo que tienen mayor movilidad, y consecuentemente mayor rango de movimientos.

Por otro lado, tener la oportunidad de desarrollar el proyecto de forma no solamente teórica sino también práctica, y además en el grupo de investigación ICARUS, fueron otras de las razones que motivaron e impulsaron la realización del proyecto.

Tecnologías empleadas

Típicamente, para detectar el movimiento voluntario de una mano se utilizan acelerómetros [6], por lo cual se pensó en usar el reloj eZ430-Chronos de Texas Instruments, ya que tiene integrado un acelerómetro de tres ejes a partir del cual se pueden obtener los datos para realizar el monitoreo de la mano. Asimismo, al ser un dispositivo inalámbrico no limita ni interfiere, dejando así al usuario moverse libremente.

Por otro lado, el dron que se ha empleado con la finalidad de tener un demostrador es el AR.Drone 2.0 de Parrot, un cuadricóptero diseñado para volar en interiores y exteriores con una gran estabilidad. Cabe señalar, que pese a estar destinado a fines recreativos, cuenta con un paquete de desarrollo oficial de descarga libre [7].

También se ha analizado en profundidad el Crazyflie 2.0 de Bitcraze, un dron de tamaño muy pequeño, con alrededor de 9 cm^2 de superficie, ideal para interiores. Este modelo resulta de especial interés, dado que es un instrumento pensado específicamente para la investigación, de manera que posee todo su código fuente abierto, siendo así mucho más sencillo hacer cualquier tipo de prueba o modificación.

Python ha sido el lenguaje de programación utilizado para el procesado de los datos de aceleración, la detección de movimientos y el envío de comandos al dron, todo ello se ha desarrollado en el IDE Pycharm de JetBrains [8]. Se eligió dicho lenguaje, ya que además de ser multiplataforma, existen APIs escritas en Python para realizar el envío de comandos a los drones analizados, tanto para el AR.Drone 2.0 como para el Crazyflie 2.0

Planificación

Este trabajo de fin de grado se ha desarrollado durante un periodo aproximado de cinco meses, empezando en septiembre de 2016 y finalizando en enero de 2017. En el diagrama de Gantt de la Fig. 0.1 se puede ver la distribución de temporal de las principales tareas.

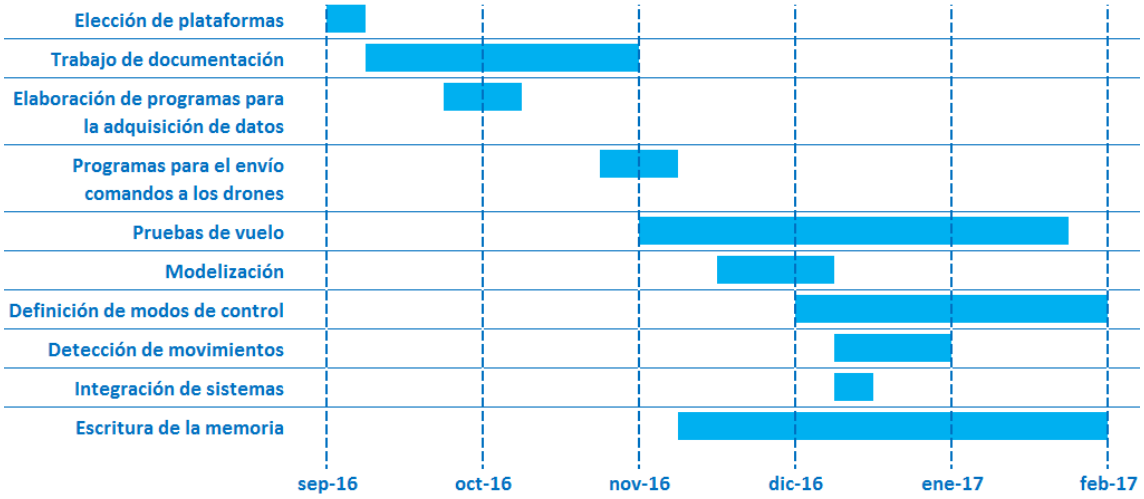

Fig. 0.1. Diagrama de Gantt

La primera semana de septiembre, se hizo la elección de las diferentes plataformas y dispositivos a utilizar, después se pasó al estudio de la documentación que ocupó más de un mes y medio. A finales de septiembre, se pasó a trabajar en la adquisición de datos del reloj, durante aproximadamente dos semanas.

Más tarde, a finales octubre y principios de noviembre se elaboraron distintos programas en Python para enviar órdenes a los drones, primero con el Crazyflie 2.0 y luego con el AR.Drone 2.0. Fue también entonces cuando se empezaron a hacer pruebas de vuelo, utilizando mandos convencionales o Python para controlarlos.

Desde finales de noviembre hasta diciembre, se trabajó en la modelización, la definición de los diversos modos de control y la detección de los movimientos del usuario. Al mismo tiempo, se hizo la integración de sistemas combinando así la adquisición de datos y el movimiento del AR.Drone 2.0 en tiempo real. Las pruebas de vuelo con el sistema completo, se empezaron a hacer entre principios y mediados de diciembre.

CAPÍTULO 1. ADQUISICIÓN DE MOVIMIENTOS

Como ya se ha adelantado en la introducción, el primer paso a seguir es obtener los datos que permitan determinar si la mano está realizando algún tipo de movimiento, y en caso afirmativo en qué dirección.

La forma convencional de captar la actividad voluntaria de una mano, es usando acelerómetros [6]. A la señal resultante se aplican filtros mediante los cuales es posible detectar inclinaciones respecto al vector de la gravedad terrestre, o aceleraciones lineales.

Con el objetivo de permitir que el usuario pueda moverse libremente sin ningún tipo de limitación u obstáculo, se decidió que el dispositivo de adquisición debía ser inalámbrico y de pequeño tamaño, además de capaz de transmitir los datos en tiempo real. Se pensó en diferentes opciones, como una unidad de medición inercial (IMU), pero finalmente se eligió el reloj eZ430-Chronos de Texas Instruments, el cual dispone de un acelerómetro de tres ejes entre otros muchos sensores.

En este capítulo se detallan las características y funcionamiento del reloj así como los diferentes métodos de adquisición de datos y el procesado que se les aplica.

1.1. eZ430-Chronos

El eZ430-Chronos es un reloj deportivo, inalámbrico y reprogramable basado en el chip CC430F6137 [9], por lo que también se puede considerar como una interesante herramienta de desarrollo. Tiene integrada una pantalla LCD de 96 segmentos como puede observarse en la Fig. 1.1 (a), además de distintos sensores para medir presión, temperatura, voltaje y aceleración en los tres ejes. Aunque para el propósito de este proyecto es necesario obtener únicamente los datos de aceleración.

(a) Reloj eZ430 (b) RF Access Point

Fig. 1.1. Componentes del eZ430-Chronos de Texas Instruments

1.2. Obtención de los datos de aceleración

El reloj eZ430-Chronos, viene con un RF Access Point que se puede ver en la Fig. 1.1 (b), el cual permite el envío de datos del reloj al PC y viceversa. Para iniciar la comunicación los pasos a seguir son:

1. Insertar en un puerto USB disponible.
2. Enviar el paquete inicial de 3 bytes: [255, 7, 3].
3. Pulsar # varias veces hasta que aparezca la palabra ACC en pantalla.
4. Pulsar ▽.

En el paquete inicial que se envía, el primer byte indica el inicio del mismo, el segundo el comando de inicio de comunicación y el tercero la longitud del paquete. Cuando se completan todos los pasos mencionados, la línea de comunicación queda iniciada, de forma que se pueden empezar a transmitir los paquetes de petición de datos.

El mensaje asociado a la petición de datos es: [255, 8, 7, 0, 0, 0, 0], y como en el caso anterior el primer byte indica el comienzo del paquete, el segundo el comando, en este caso la petición de datos y el tercero la longitud del paquete enviado. Una vez se envía, el reloj contesta a dicha petición con un paquete también de 7 bytes, de los cuales los tres primeros son cabeceras, el cuarto byte indica la disponibilidad de los datos (1 si los datos están disponibles y 255 en caso contrario) mientras que los últimos 3 bytes son las aceleraciones en los ejes X, Y, Z respectivamente.

En la siguiente imagen, Fig. 1.2 se adjunta un diagrama con el proceso a seguir para la obtención de datos.

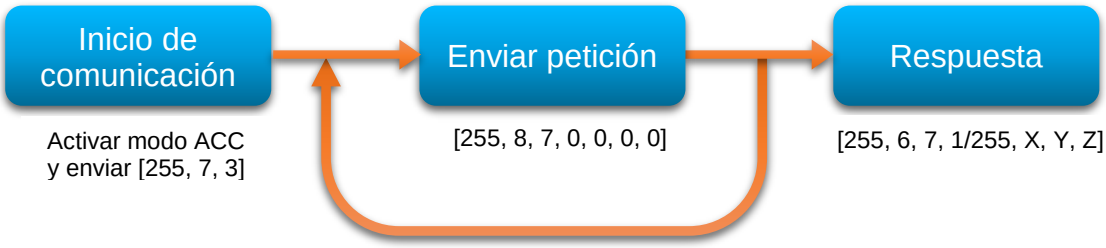

Fig. 1.2. Diagrama de adquisición de datos

Una vez finalizada la adquisición de datos, para que la comunicación quede interrumpida, se transmite el siguiente paquete de 3 bytes: [255, 9, 3], de nuevo el primer byte hace referencia a que se trata del comienzo del paquete, el segundo al comando de interrupción de comunicación y el tercero a la longitud del paquete. De todos modos, también es posible parar la comunicación pulsando el botón ▽ del reloj.

La información referente a los paquetes que es necesario enviar en cada momento, se obtuvo a partir del análisis de un programa en LabVIEW [10], adjunto a la información proporcionada por Texas Instruments, dónde se mostraban los datos de aceleración de cada eje en varios gráficos.

Dada la escasa documentación, para verificar que se estaban enviando los paquetes correctos, se hizo un pequeño código en C# que leía las aceleraciones medidas por el eZ430-Chronos. En el Anexo I se encuentra el programa, junto con una breve explicación de sus parámetros y funcionamiento.

Haciendo pruebas modificando el tiempo de envío de peticiones al reloj, se concluyó que la frecuencia máxima a la cual puede responder el eZ430-Chronos es de 23 Hz, o lo que es lo mismo cada 45 ms. En la Fig. 1.3, puede verse un gráfico que muestra el número de paquetes recibidos por minuto, cuando se muestrea a dicha frecuencia.

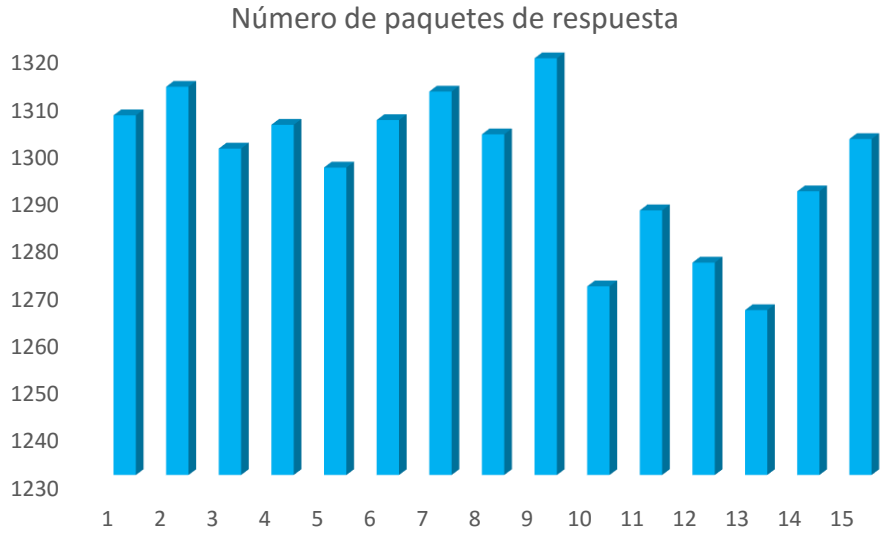

Fig. 1.3. Número de paquetes por minuto con tiempo de muestreo de 45 ms

Como se observa a partir del gráfico anterior, el número máximo de respuestas es de media 1295 paquetes por minuto aproximadamente. Con lo cual se pudo confirmar que el dispositivo elegido, transmite a una velocidad más que suficiente para la aplicación que se pretende diseñar.

1.3. Diseño e implementación del sistema de adquisición

Los datos de aceleración que entrega el reloj, no son más que enteros que van de 0 a 255. Es importante tener en cuenta que los valores que van desde 0 hasta 127 corresponden a aceleraciones en un sentido, y de 128 a 255 al sentido contrario. Esta regla es aplicable a los tres ejes del acelerómetro.

Si a esto le sumamos el hecho de que no todos los datos que proporciona el reloj son necesariamente significativos, ya que en ocasiones se reciben cambios muy pequeños respecto a los valores anteriores y se genera ruido, resulta imprescindible tratar los datos.

Una vez se reciben los paquetes de respuesta del reloj, los cuales contienen los datos de aceleración, es necesario aplicar un preprocesado para determinar cuál es el sentido de la aceleración medida. De manera, que se pueda así escalar los valores y asignar un signo positivo o negativo.

Después del preprocesado, es también esencial aplicar un filtro para ignorar los cambios de valores no significativos, es decir, los cambios que son muy pequeños, y así obtener una posición estable.

Asimismo, se han definido tres modos diferentes de filtrado de aceleraciones para limitar o restringir los valores. La razón principal de la creación de distintos modos, fue querer facilitar la integración de sistemas más adelante, así como facilitar también el pilotaje del vehículo aéreo no tripulado al usuario del sistema. En el Anexo II, se adjuntan los algoritmos utilizados para el preprocesado y filtrado de valores.

A continuación se enumeran los distintos modos, explicando también las características de cada uno de ellos.

- **Modo libre.**
  Permite el movimiento en cualquier dirección sin ningún tipo de restricción.

- **Modo limitado a un eje.**
  Considera el movimiento solamente en el eje en el cual hay una mayor diferencia de aceleraciones, mientras que el eje restante se mantiene constante en la posición inicial.

- **Modo deltas.**
  Tiene en cuenta el movimiento en un único eje, como en el caso anterior, pero solo cuando se sobrepasa un cierto umbral predefinido, es decir, una vez detecta un cambio de aceleración muy significativo dado por un movimiento de amplio rango en un eje.

Se han realizado dos programas, uno en LabVIEW y otro en Python, ambos realizan las mismas funciones y aplican los mismos tipos de filtrado. En un primer momento se hizo el programa en LabVIEW para reducir tiempo, ya que se aprovechó parte del código de la documentación del reloj [10].

Más tarde, se hizo el mismo programa en Python, dada la existencia de APIs escritas en Python para realizar el envío de comandos a los drones con los que se trabajaría más adelante. Además, a diferencia de LabVIEW, Python se trata de un lenguaje multiplataforma adaptable a cualquier sistema.

En la Fig. 1.4 puede verse la secuencia de funciones de los programas en LabVIEW y Python. Es válido para los tres modos, ya que únicamente cambia el filtrado que se aplica en cada caso.

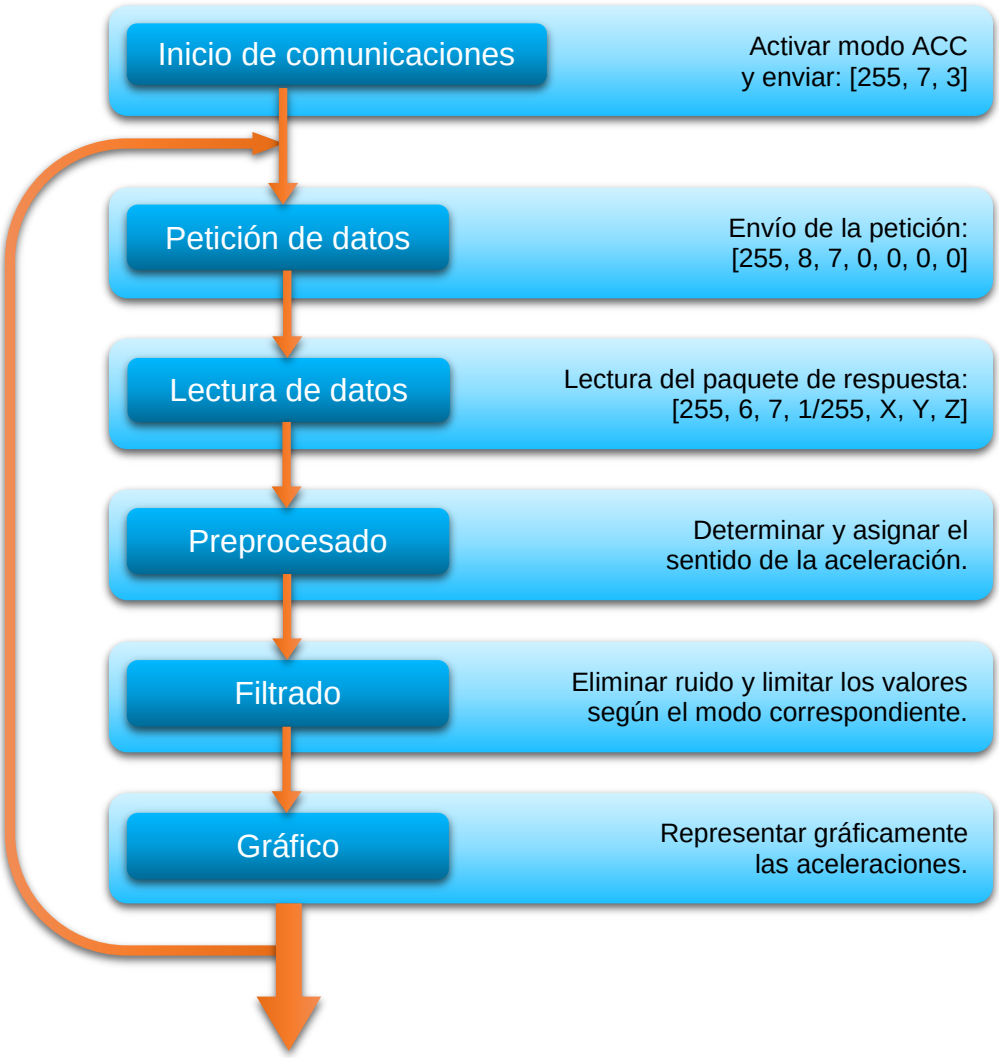

Fig. 1.4. Diagrama secuencial de funciones de los programas

1.3.1. Adquisición de datos mediante LabVIEW

LabVIEW es un software de National Instruments [11], pensado para el desarrollo con una sintaxis de programación gráfica que facilita visualizar, crear y codificar sistemas de ingeniería, por lo que suele ayudar a reducir tiempos de pruebas.

Tal y como ya se ha comentado en el anterior punto, en la documentación de Texas Instruments había adjunto un código en LabVIEW [10]. Dicho programa originalmente simplemente leía, preprocesaba y mostraba en función del tiempo las aceleraciones medidas por el reloj. Más tarde, se modificó añadiendo un filtrado de valores de las aceleraciones en los ejes X e Y para eliminar el ruido.

Una vez se ajustaron los filtros y se consiguieron posiciones más estables, se hicieron otras dos versiones del mismo programa añadiendo más filtros y restricciones con la finalidad de implementar el modo limitado a un eje y el modo deltas.

A continuación, en la Fig. 1.5 se encuentra la interfaz gráfica original del programa de la documentación de Texas Instruments. En la primera, segunda y tercera gráfica, puede verse la representación ejes X, Y, Z en función del tiempo respectivamente. En la cuarta aparece la correlación de los ejes X e Y.

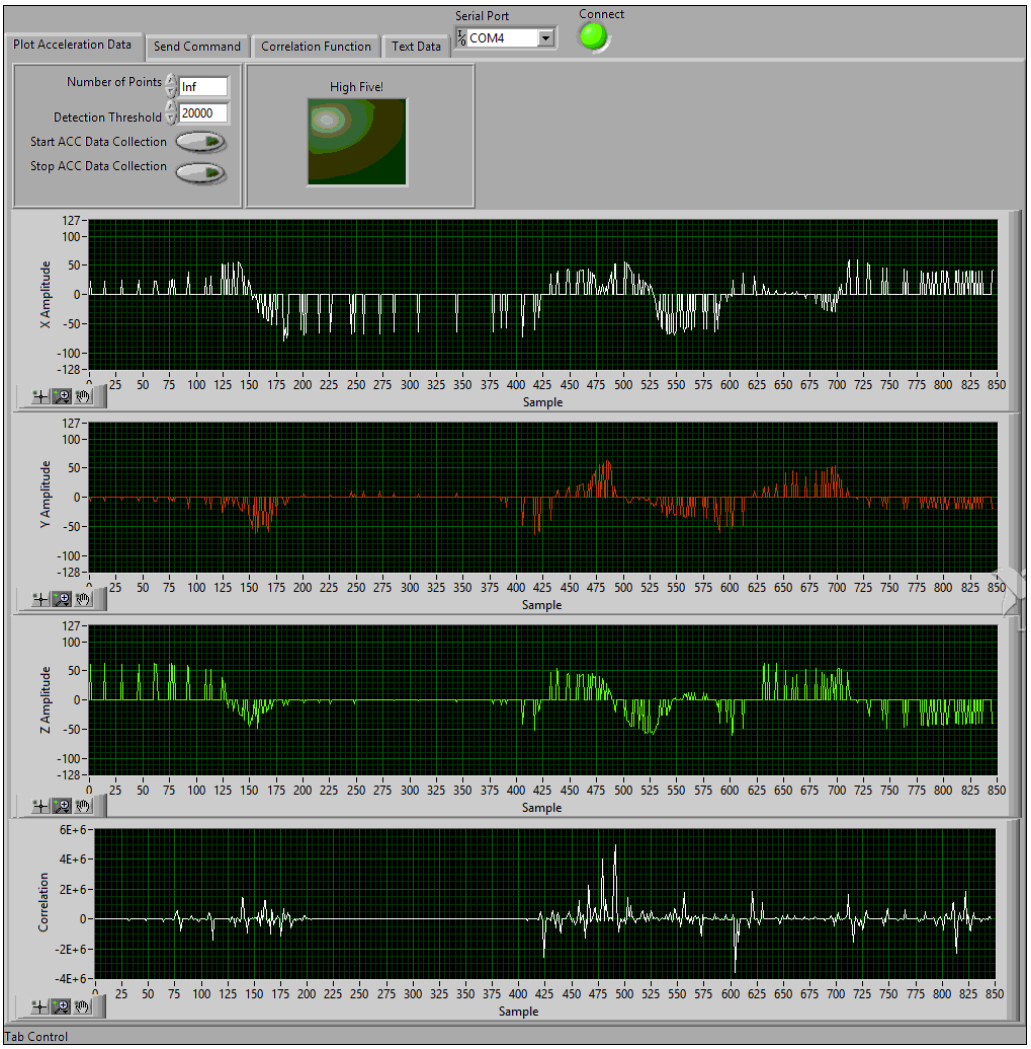

Fig. 1. 5. Interfaz original del programa de la documentación de Texas Instruments

Por otra parte, en la Fig. 1.6 se encuentra la interfaz gráfica del programa modificado, tal y como ya se ha comentado al programa original se añadieron filtros para eliminar el ruido e implementar los distintos modos. Como puede observarse, los tres primeros gráficos corresponden a las aceleraciones en función del tiempo en los ejes X, Y, Z respectivamente y el último a la trayectoria.

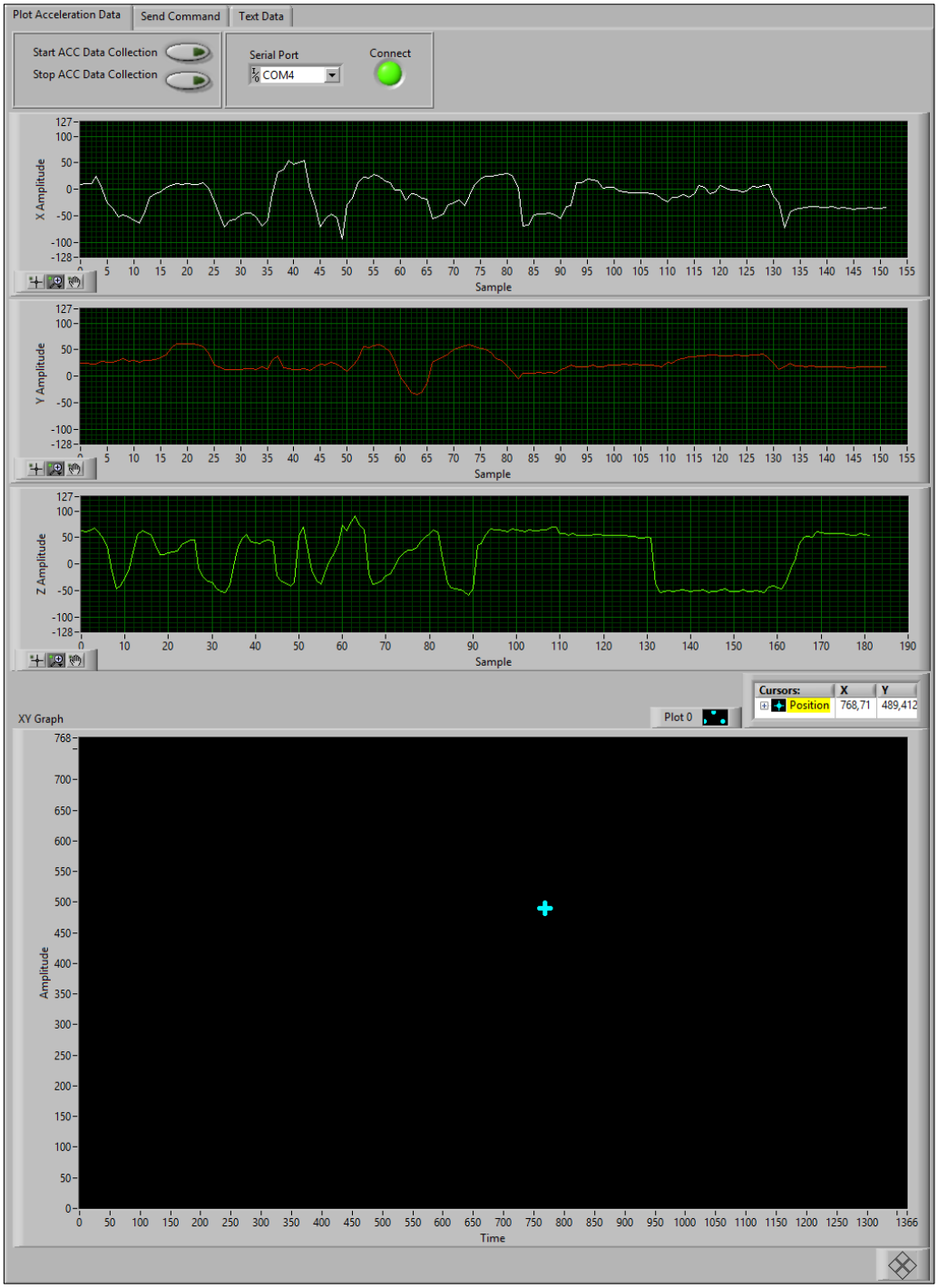

Fig. 1.6. Interfaz del programa después de las modificaciones

1.3.2. Adquisición de datos mediante Python

Python es un lenguaje de programación que cuenta con estructuras de datos eficientes y de alto nivel [12], además de un enfoque simple pero efectivo a la programación orientada a objetos. El motivo de replicar el código ya hecho en LabVIEW es la existencia de APIs escritas en Python, que permiten el envío de comandos a los drones elegidos para trabajar.

Asimismo, Python es un lenguaje multiplataforma por lo que proporciona una mejor adaptabilidad a cualquier sistema. De la misma forma que en LabVIEW, se han programado los tres diferentes modos limitadores, siguiendo estrictamente la guía de estilo PEP 8 [13], con el objetivo de mejorar la legibilidad del código.

En la siguiente imagen, la Fig. 1.7 puede observarse un diagrama con la secuencia de las distintas clases, métodos y atributos que conforman el código del modo libre. En primer lugar, como se ha explicado anteriormente se establece la comunicación con el puerto serie, se envía la petición de datos y se lee la respuesta. Después se realiza el filtrado y en caso de que la variable booleana, *save_into_file*, sea igual a *True* se guardan los datos filtrados en un fichero de texto. Si es igual a *False* se pasa directamente a actualizar el dato en pantalla, en un gráfico de trayectoria a partir de las aceleraciones X e Y medidas por el reloj.

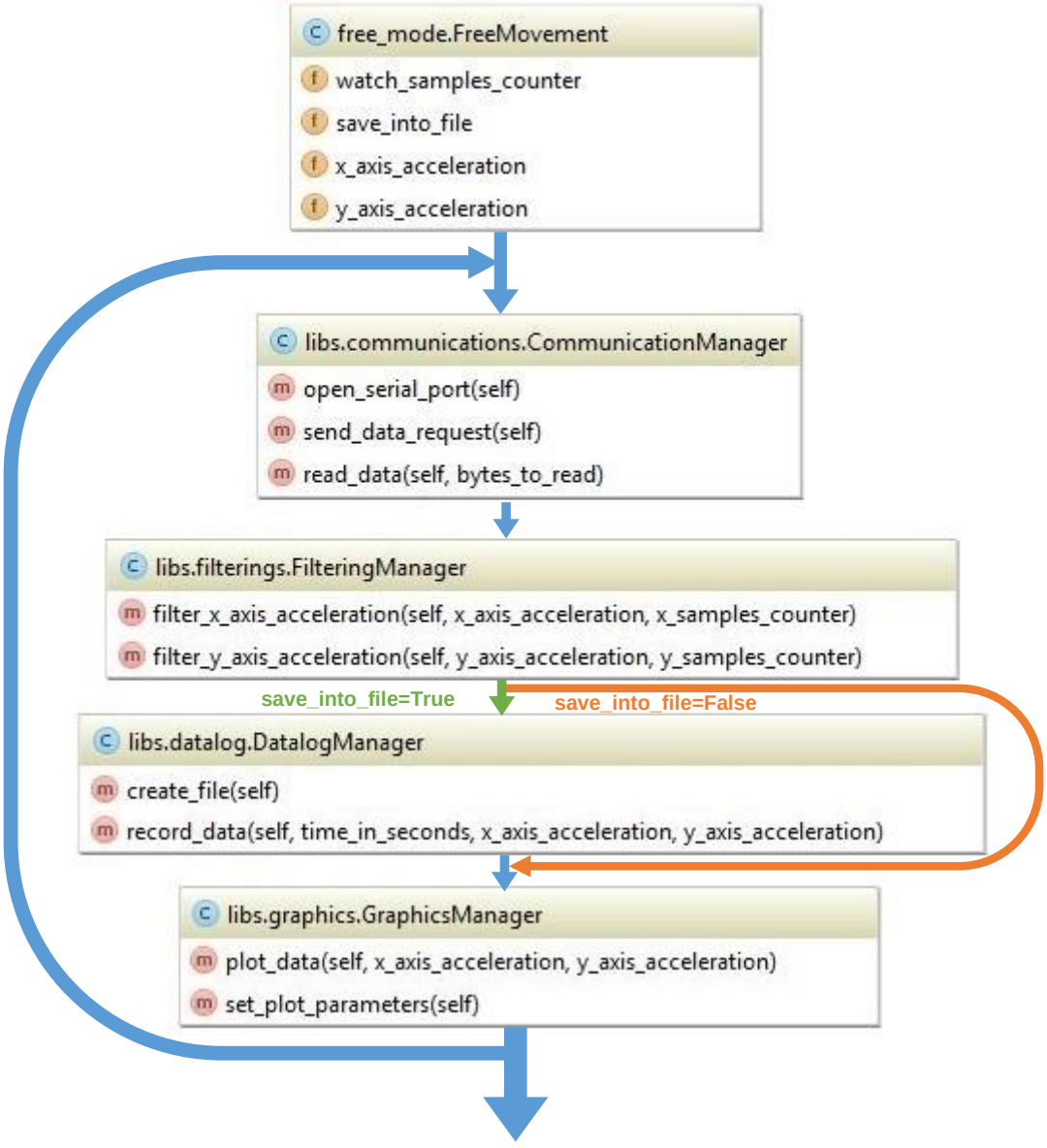

Fig. 1.7. Diagrama secuencial del modo libre

Seguidamente, en la Fig. 1.8 se puede ver la función del modo limitado a un eje. El diagrama secuencial es igual al precedente, el caso del modo libre, la única diferencia es que después del filtrado, se añade un segundo filtro que verifica en qué eje es mayor la diferencia de aceleraciones y limita el movimiento en el otro manteniéndolo constante en la posición inicial.

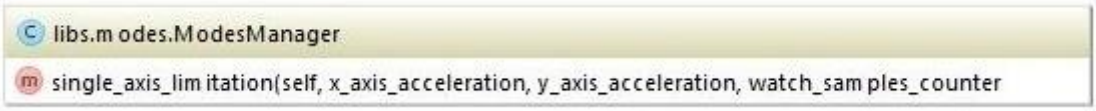

Fig. 1.8. Diagrama secuencial del modo limitado a un eje

Por último, la Fig. 1.9 muestra la función del modo deltas. Como en el caso anterior, solo se diferencia del modo libre en que se añade un segundo filtro que tiene en cuenta únicamente los movimientos bruscos, que están por encima de un umbral definido.

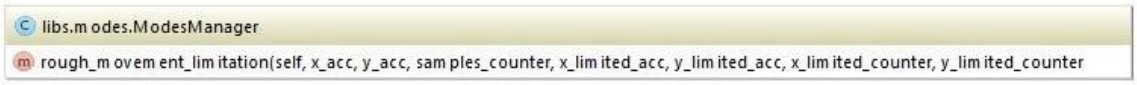

Fig. 1.9. Diagrama secuencial del modo deltas

1.3.3. Comparativa de la adquisición con LabVIEW y Python

Una vez hechos los dos programas, se ha querido comprobar la correspondencia de datos que hay entre ambos con el objetivo de verificar que el rendimiento de ambos programas fuera el mismo. Para ello se han ejecutado los dos programas al mismo tiempo durante un minuto y posteriormente se ha guardado un registro de las posiciones mapeadas en un fichero de texto.

Mediante el programa Virtual Serial Ports Emulator se ha creado un *splitter*, es decir, que a partir del puerto serie donde se conecta el RF Access Point se ha creado un puerto serie virtual, teniendo así la posibilidad de conectar varios programas en el mismo puerto, tal y como muestra el esquema de la Fig. 1.10.

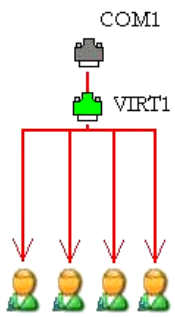

Fig. 1.10. Esquema de un *splitter* genérico

Una vez se han obtenido los registros, se han hecho los gráficos que aparecen en las Fig. 1.11 y 1.12. Se puede observar que las curvas de datos obtenidos en LabVIEW y Python son prácticamente iguales, aunque en algunos puntos puede apreciarse una cierta diferencia en el dominio temporal, ya que el tiempo de recepción de los paquetes de respuesta es muy similar en ambos programas pero no idéntico, de ahí que haya un pequeño descuadre en algunas zonas.

En todo caso, no es una diferencia significativa pues se trata de tan solo 0.8 segundos en el peor de los casos. Por otra parte en términos posición en X e Y se puede ver claramente que no hay cambios entre los datos de LabVIEW y Python.

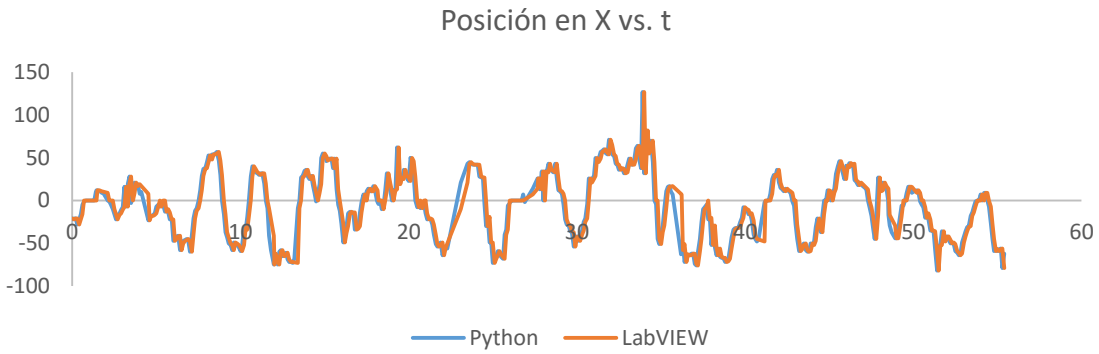

Fig. 1.11. Comparativa entre LabVIEW y Python en el eje X

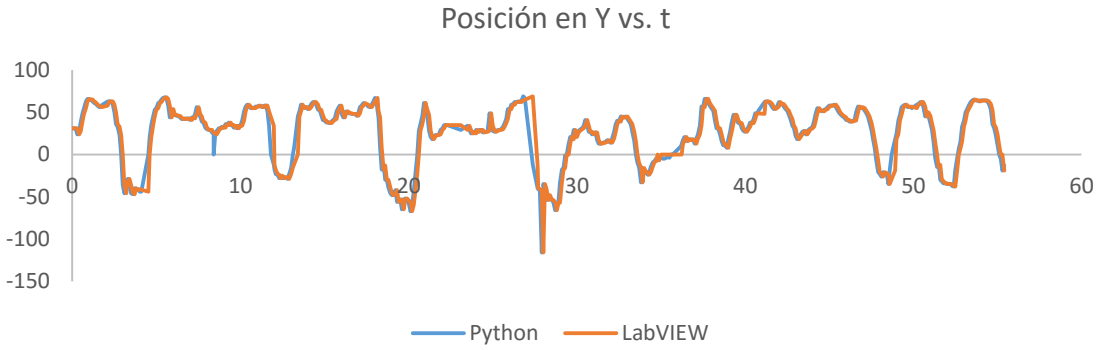

Fig. 1.12. Comparativa entre LabVIEW y Python en el eje

CAPÍTULO 2. VEHÍCULO AÉREO NO TRIPULADO

Una vez explicado el método de adquisición de las aceleraciones del eZ430-Chronos, en este capítulo se exponen las características, funcionamiento y métodos de comunicación de las dos plataformas de vehículos aéreos no tripulados con los que se ha trabajado. Por un lado, el Crazyflie 2.0 y seguidamente el AR.Drone 2.0.

2.1. Crazyflie 2.0

El Crazyflie 2.0 de Bitcraze [14], es un cuadricóptero de tamaño muy pequeño, ideal para volar en interiores y pensado específicamente para la investigación y el desarrollo, puede verse en la Fig. 2.1. La razón principal por la que se empezó a trabajar con este dron es que se trata de una plataforma de código abierto, con el diseño de hardware y el código fuente ambos documentados y enteramente disponibles.

Fig. 2.1. Crazyflie 2.0 de Bitcraze

2.1.1. Características

Con un peso de 27 g y unas medidas de 92x92 mm, su arquitectura está basada en dos microcontroladores: el NRF51, Cortex-M0 y el STM32F405, Cortex-M4@160MHz [15]. El primero se usa para gestionar las comunicaciones con la radio, el Bluetooth y el control de la alimentación, mientras que el segundo se encarga del control de vuelo y motores, la telemetría y la lectura de los sensores.

Cuenta a su vez con una memoria EEPROM de 8 KB para guardar parámetros de configuración, además de una unidad de medición inercial (IMU) de 10 grados de libertad ya que lleva incorporados un MPU-9250, el cual contiene acelerómetro, giróscopo y magnetómetro todos de 3 ejes; y un sensor LPS25H, que hace mediciones de presión con alta precisión.

En relación a la autonomía, el tiempo de vuelo medio es de unos 7 minutos con una batería LiPo de 240 mAh y sin carga de pago. El peso máximo de despegue es de unos 42 g, lo que supone una carga de pago de 15 g y una disminución considerable del tiempo de vuelo.

Para más información sobre las características, consultar el Anexo III donde se encuentra documentación más detallada.

2.1.2. Modelo de comunicaciones

El diagrama de la Fig. 2.2 muestra de las distintas formas que existen para comunicarse con el Crazyflie 2.0. Tal y como se aprecia, el cuadricóptero puede ser controlado desde un ordenador con cualquier mando que tenga un mínimo de cuatro ejes analógicos, mediante un enlace RF. También es posible controlarlo con Bluetooth a través de un smartphone, utilizando la aplicación creada por Bitcraze, disponible para Android [16] e iOS [17].

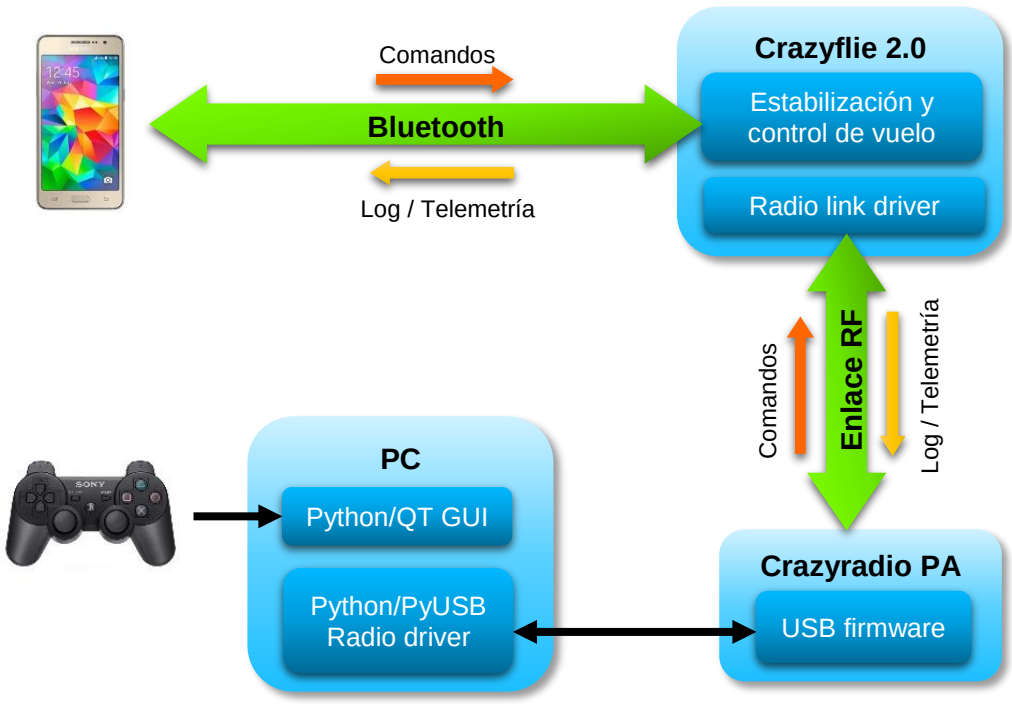

Fig. 2.2. Modelo de comunicaciones del Crazyflie 2.0

El control a través del teléfono móvil es quizás la manera más rápida de ponerlo en marcha, pero requiere una mayor soltura a la hora de pilotar, ya que los controles táctiles no son tan fáciles de utilizar como un mando convencional. Para la elaboración de este proyecto, resulta más interesante el control a partir del PC, dado que es donde está basado el sistema de adquisición de movimientos, es decir, donde se recibe los datos del reloj eZ430-Chronos.

Tradicionalmente, para realizar el control desde el PC, se utiliza una máquina virtual creada por Bitcraze. Contiene todas las dependencias y extensiones necesarias instaladas, junto con un cliente que posee una interfaz gráfica de usuario (GUI). De todos modos, el cliente puede ser instalado en los sistemas operativos nativos, está disponible para Windows, Mac OSX y Linux [18].

Dicho cliente, se comunica mediante *cflib*, una interfaz de programación de aplicaciones (API) escrita en Python. Además, se requiere la Crazyradio PA que puede verse en la Fig. 2.3, y un mando con un mínimo de cuatro ejes para pilotarlo.

Fig. 2.3. Crazyradio PA

Así mismo, usando la API comentada anteriormente *cflib*, es posible controlar el dron sin necesidad de mando, enviándole al dron distintos comandos a partir de un simple código en Python. En el Anexo III se adjunta un pequeño programa de ejemplo.

Los parámetros de control a tener en cuenta son cuatro: los ángulos de guiñada, alabeo, cabeceo y el empuje de los motores; en función de la maniobra que desee realizarse será necesaria una combinación de valores u otra.

2.1.3. Pruebas de vuelo

Después de varios ensayos y pruebas de vuelo, tanto con mando como mediante códigos en Python, se han constatado ciertas evidencias:

- Si la batería no se encuentra perfectamente centrada, el cuadricóptero es prácticamente imposible de controlar, ya que esta tiene un peso de 7.1 g, lo que supone un 26% del total, luego su posicionamiento tiene una gran influencia sobre el centro de masas del dron.

- Sin carga de pago, los motores tienen demasiada potencia con lo cual si no se compensa correctamente, de nuevo resulta difícil de controlar.

- Cuando se envían comandos a través de la máquina virtual de Bitcraze, hay una gran latencia en la respuesta del dron, lo cual lleva a que el vuelo sea altamente inestable, dado que el sistema es demasiado lento como para hacer correcciones o compensar de manera instantánea.

- El modo de altitud constante no es estable, debido a problemas de conectividad [19], el nivel de batería oscila hecho que produce oscilaciones en el valor del empuje de los motores, ya que son directamente proporcionales como muestra la Fig. 2.4 [20].

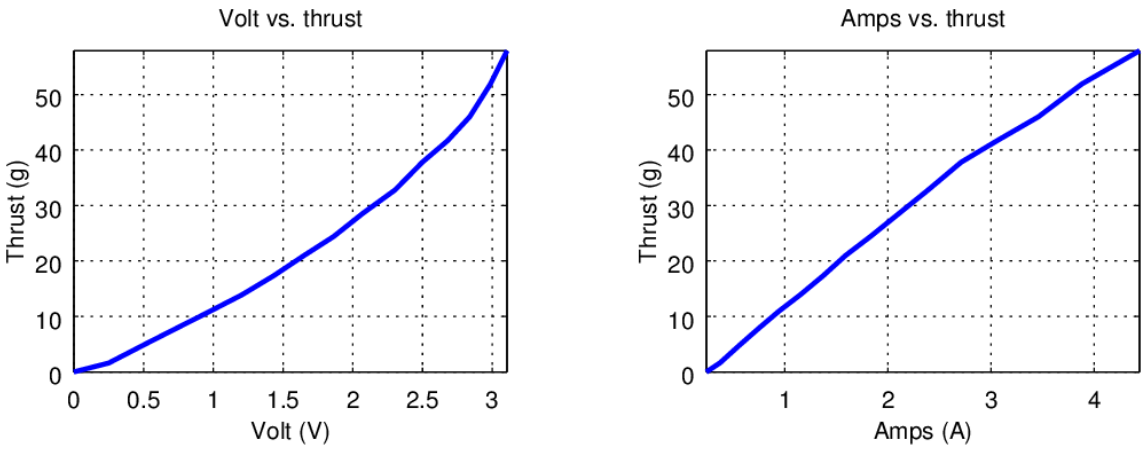

Fig. 2.4. Empuje de los motores en función del voltaje y la intensidad

- Controlándolo desde el ordenador con el cliente, el rango del mando no da demasiado margen de movimientos. Se alcanza muy rápidamente los máximos, hecho que una vez más produce inestabilidad en el vuelo.

Una vez detectados todos estos problemas de estabilidad, se ha tratado de buscar algunas mejoras que aporten una mayor precisión al vuelo del cuadricóptero.

Utilizando una impresora 3D se ha creado una estructura de protección que puede observarse en la Fig. 2.5 y que añade unos 5 g de peso al dron, haciendo así que su control sea mucho más manejable. No obstante, está solución también comporta puntos negativos, ya que se reduce el tiempo de vuelo considerablemente.

Sin estructura de protección, ni carga de pago, y considerando una batería LiPo de 240 mAh, la autonomía del cuadricóptero es de unos 7 minutos, tal y como ya se ha comentado anteriormente. Si se añade la estructura el tiempo máximo de vuelo pasa a ser 4 minutos.

Existe también un sistema de posicionamiento local diseñado por Bitcraze, que otorga una gran precisión y estabilidad. En la sección 5. Conclusiones y trabajo futuro, se ampliará más la información respecto a este sistema de posicionamiento.

Fig. 2.5. Crazyflie 2.0 con estructura de protección

2.2. AR.Drone 2.0

A causa de todos los problemas de estabilidad observados en el Crazyflie 2.0 comentados en el apartado anterior (2.1.3. Pruebas de vuelo), se optó por cambiar la plataforma de vehículo aéreo no tripulado, dado que para la realización del proyecto se necesita una mayor precisión en los movimientos. Así pues, se eligió el AR.Drone 2.0 de Parrot que puede verse en la Fig. 2.6.

Sus capacidades de vuelo no fueron el único motivo para escogerlo, otro punto a favor fue la existencia de un kit orientado al desarrollo de software (SDK) muy funcional y adaptable, sin necesidad de tener que añadir plataformas con microcontroladores como por ejemplo Arduino.

2.2.1. Características

El AR.Drone 2.0 es un cuadricóptero de tamaño mucho mayor al Crazyflie 2.0, sin embargo está diseñado para vuelo tanto en interiores como exteriores, tal y como muestra la Fig. 2.6 hay unas protecciones específicas para vuelo en interiores y exteriores, que suponen unas medidas de 525x515 mm y 490x290 mm respectivamente.

Fig. 2.6. AR.Drone 2.0 de Parrot con protección para vuelo en interiores (izquierda) y en exteriores (derecha)

Su estructura es muy ligera, el peso total con batería y carcasa de interiores es de 465 g, siendo la parte central la más pesada ya que es donde se encuentra el cuerpo que alberga los elementos más significativos del dispositivo.

Contiene un microprocesador ARM Cortex A8 de 32 bits, una memoria DDR2 RAM de 1 GB @200MHz [21], un sistema operativo con núcleo Linux versión 2.6.32.9 (BusyBox), un modem Wi-Fi b/g y además un conector USB de alta velocidad para poder almacenar los vídeos y las fotos que se realicen.

En cuanto su precisión, está altamente equipado pues cuenta con acelerómetro, giróscopo y magnetómetro de todos tres ejes, sensor de presión, sensor de ultrasonidos para medir la altitud y cámara vertical para medir la velocidad en vuelo. En referencia a la autonomía, teniendo en cuenta que la carcasa de exteriores pesa el tiempo de vuelo es de unos 12 minutos aproximadamente utilizando una batería LiPo de 1500 mAh. Para más información sobre las características, consultar el Anexo IV donde se encuentra información más detallada.

2.2.2. Modelo de comunicaciones

Este dron está concebido para controlarse con el teléfono móvil mediante la aplicación creada por Parrot, disponible para Android [22] e iOS [23].

Tal y como muestra la Fig. 2.7, el *framework* interconecta el controlador visual con el AR.Drone 2.0 a través la API, la red, el *proxy* y el SDK de Parrot. El sistema total sigue una estructura de control en cascada. Aun así, hay bucles de control de alta frecuencia a bordo para la regulación de la velocidad de los motores y la actitud del cuadricóptero. El controlador visual se cierra en un nivel superior a una frecuencia menor [24].

La red consta de cuatro canales para vídeo, datos de navegación, envío de comandos y envío de datos críticos, como los referentes a la configuración. Los tres primeros canales mencionados son puertos UDP mientras que el último es TCP para asegurar el envío de datos. A continuación se profundiza más en cada uno de los canales.

- **Canal de vídeo. Puerto UDP 5555.**
  El dron lo usa para transferir de vídeo de forma continua al cliente.

- **Canal de navdata. Puerto UDP 5554**
  El AR.Drone lo utiliza para enviar de datos de navegación como: estado del dron, posición, velocidad, velocidad de rotación de los motores, etc.

- **Canal de comandos. Puerto UDP 5556.**
  La aplicación de control, es decir, el cliente lo usa para enviar los comandos con la finalidad de poder controlar el dron.

- **Canal de control. Puerto TCP 5559.**
  El cliente lo utiliza para enviar al dron datos críticos de configuración.

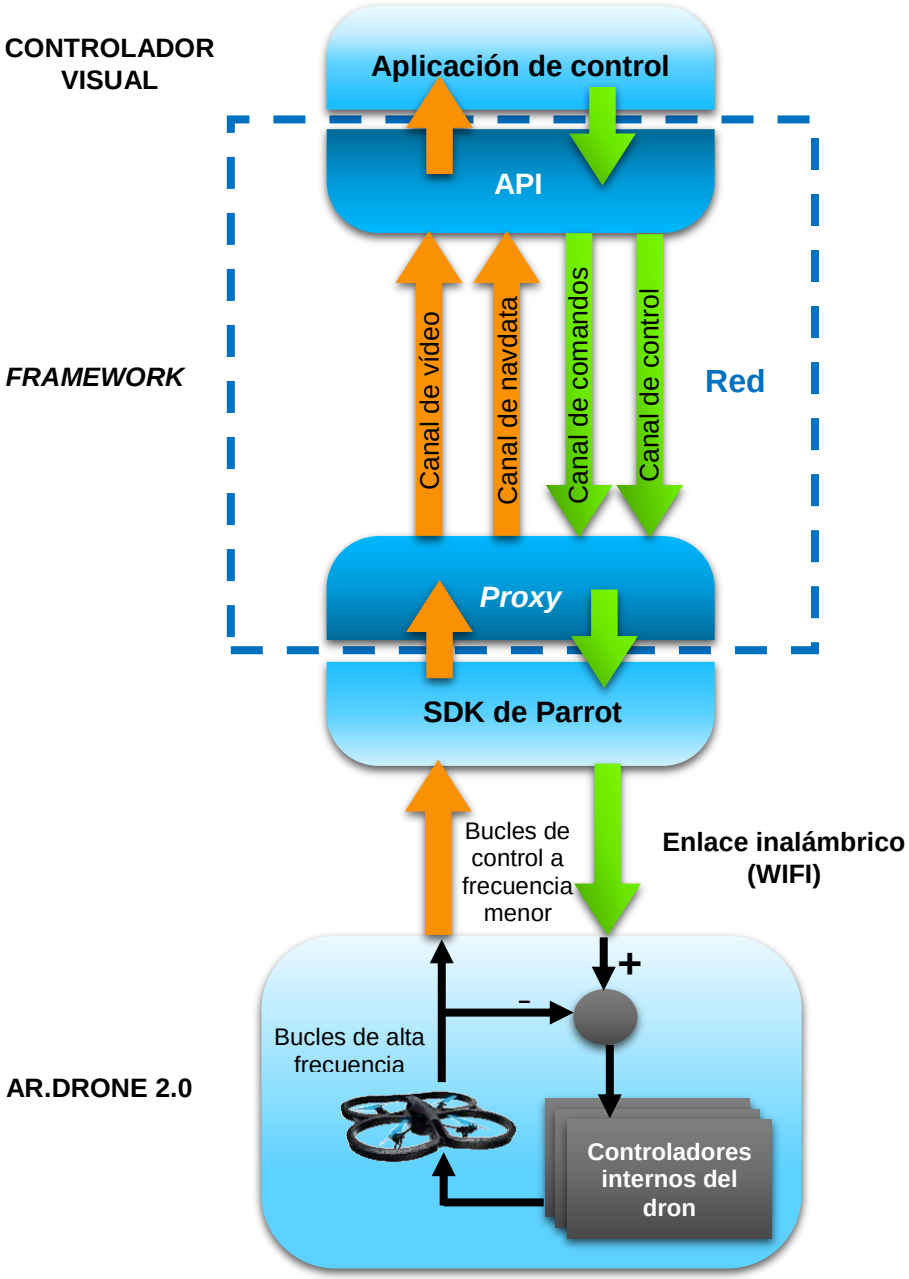

Fig. 2.7. Modelo de comunicaciones del AR.Drone 2.0

Como ya se ha comentado previamente en secciones anteriores, para la elaboración de este proyecto nos interesa el control del vehículo aéreo no tripulado a partir del ordenador, ya que es donde está basado el sistema de adquisición de datos.

A diferencia de Bitcraze, Parrot no ha elaborado ninguna API, pese a eso existen múltiples bibliotecas no oficiales creadas por algunos usuarios a partir del SDK.

Para interactuar con el dron, se ha utilizado la biblioteca *pyardrone* [25] escrita en Python, que hace posible el envío y recepción de paquetes de datos al AR.Drone 2.0. Incorpora una API de alto nivel con comandos básicos tales como

despegue, aterrizaje, y movimientos en cualquier dirección. Además se han añadido comandos que no estaban en un primer momento, afín de compensar el dron y activar o desactivar el modo emergencia. Dicho modo consiste en el corte de subministro eléctrico a los motores y suele activarse en situaciones críticas, como por ejemplo cuando se detecta una inclinación excesiva.

En el Anexo IV se adjunta un pequeño programa de ejemplo donde se usa la biblioteca *pyardrone* para controlar el vuelo del AR.Drone 2.0.

2.2.3. Pruebas de vuelo

Una vez probados los dos cuadricóptero que se han presentado en este capítulo, se llegó a la conclusión de que el AR.Drone 2.0 es una plataforma mucho más adecuado para el fin que se persigue en este proyecto.

La cámara ventral lo dota de una gran precisión, haciendo su vuelo estable incluso en exteriores donde interviene una cierta componente del viento. La cámara, capta imágenes de su ubicación inicial y las utiliza para crear un sistema de posicionamiento.

Se ha observado, que poniendo una imagen asimétrica y de alto contraste como la que aparece en la Fig. 2.8, justo debajo del dron antes del despegue, se obtiene más precisión si cabe. Por otro lado, si la superficie en la que se realiza el vuelo es muy homogénea, el dron puede confundir su ubicación inicial y desviarse de la trayectoria correcta.

Fig. 2.8. Imagen asimétrica de ayuda para el posicionamiento

De forma experimental, después de múltiples pruebas de vuelo, se consiguieron determinar los valores necesarios para controlar el desplazamiento real del dron, tanto en el plano longitudinal como en altitud. Los dos parámetros a tener en cuenta al realizar el envío de comandos mediante la API, son la velocidad del movimiento y el su tiempo de duración.

Las velocidades de cada comando de movimiento, se expresan con porcentajes de la velocidad máxima del dron, que es de 5 m/s, es decir, 18 km/h. Por otra parte, el tiempo se expresa en segundos.

A continuación, se incluyen los valores necesarios para obtener un desplazamiento real de un metro con el AR.Drone 2.0. Notar que para un cambio de altitud, es decir, un desplazamiento vertical, se necesita más velocidad que en el caso de un desplazamiento longitudinal. A diferencia de este último, en un desplazamiento vertical interviene la componente del peso del dron, de ahí que sea necesaria una mayor velocidad.

- **Desplazamiento longitudinal de 1 metro.**
  Velocidad: 10%
  Tiempo: 2 segundos

- **Desplazamiento vertical de 1 metro.**
  Velocidad: 30%
  Tiempo: 2 segundos

A parte de su estabilidad longitudinal, es capaz de mantener altitud constante sin problemas, por todo ello finalmente se descartó el Crazyflie 2.0 y se pasó a trabajar únicamente con el AR.Drone 2.0.

CAPÍTULO 3. MODELIZACIÓN

En esta sección se trata la modelización que se ha establecido para traducir los movimientos de la mano del usuario, a órdenes para el cuadricóptero. Se detallan los diversos métodos empleados para la detección de movimientos y la traducción de estos a comandos para el AR.Drone 2.0, además del modo de control que se ha definido para pilotar mediante el reloj eZ430-Chronos.

3.1. Estudio de los movimientos

El primer paso fue hacer un estudio de los patrones que seguían distintos movimientos, después de múltiples pruebas se concluyó que los cambios más significativos se producían al hacer giros de muñeca, en lugar de moviendo el brazo entero.

La razón por la cual se pueden detectar más fácilmente giros de muñeca que movimientos de translación, es que con los giros el reloj se inclina. Cuando se produce una inclinación, la gravedad genera componentes vectoriales en los ejes X, Y, Z del acelerómetro del eZ430-Chronos. Estas componentes, tienen un valor mucho mayor a cualquier aceleración lineal que pueda efectuar físicamente una persona con su brazo.

Seguidamente, se definieron las dos posiciones de equilibrio que aparecen a continuación en la Fig. 3.1, siempre manteniendo el brazo flexionado en un ángulo de 90º. A partir de estas posiciones, se ejecutan los movimientos de interés.

Fig. 3.1. Posiciones de equilibrio

El hecho de fijar estas dos referencias, sirve para normalizar los valores y facilitar la detección de patrones, ya que la gravedad afecta de una forma u otra en función de la posición del brazo y la muñeca, por lo tanto si no se imponen unas

normas de utilización resulta prácticamente imposible la correcta identificación de movimientos. Además, la fijación de estas dos posiciones de equilibrio proporciona una mayor cantidad de comandos para poder controlar el dron. Con una posición, solo es posible detectar correctamente cuatro movimientos distintos, mientras que teniendo dos posiciones de equilibrio. Considerando que cada movimiento lleva asociado un comando distinto para el dron, con dos posiciones de equilibrio se pasa a poder enviar ocho órdenes distintas.

En un primer momento, se pensó en el chasquido como movimiento para enviar el comando de despegue al cuadricóptero, sin embargo más adelante se vieron ciertas dificultades para identificar el patrón. Únicamente se percibían cambios significativos en el eje Z, los ejes X e Y restaban impasibles, siendo así difícil asegurar la detección del patrón con un porcentaje de error bajo.

Al ser el despegue una orden de vital importancia, el hecho de usar un movimiento tan crítico como el chasquido, no era del todo fiable y podía dar lugar a errores. Con lo cual se consideró que una palmada era un movimiento más adecuado, ya que proporcionaba cambios de valores significativos en todos los ejes. De todos modos, se decidió añadir un grado de redundancia y probar con dos palmadas, ya que la repetición elimina la posibilidad de que se detecte un falso positivo.

Así pues se eligieron dos palmadas seguidas como comando de despegue partiendo de la posición de equilibrio 1. Su patrón se puede observar en la siguiente Fig. 3.2, de donde se concluye que pese que hay información relevante en todos los ejes, en el eje Z se obtiene un cambio mucho más significativo que en los demás.

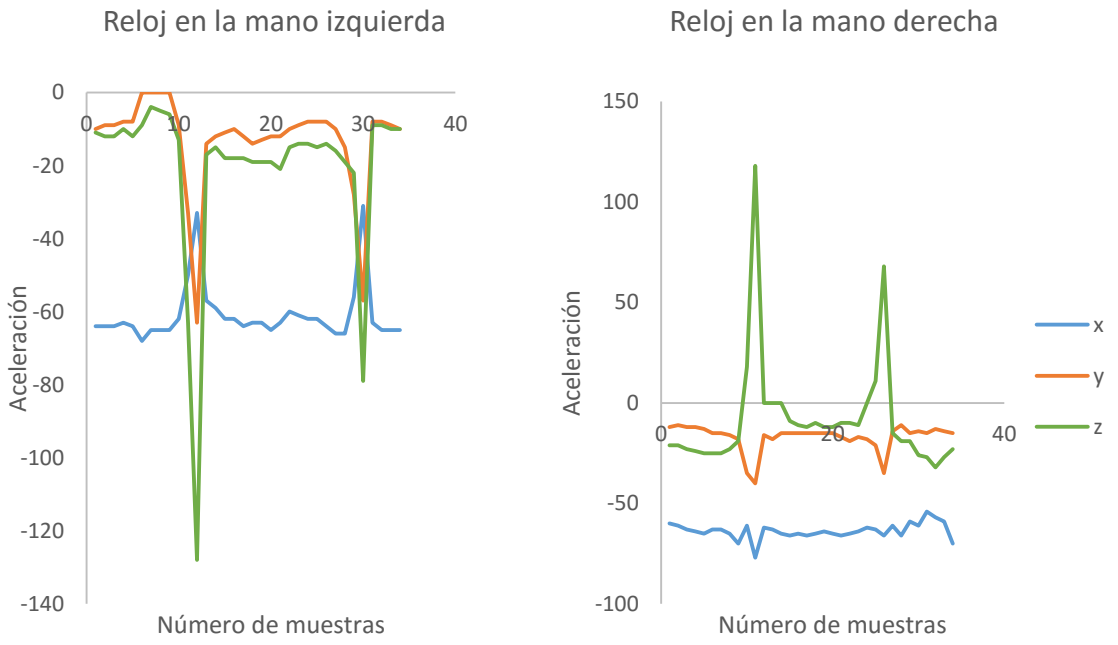

Fig. 3.2. Patrón de dos palmadas

Partiendo de nuevo de la posición de equilibrio 1, al realizarse giros de muñeca en dirección izquierda y derecha, se encontraron las tendencias que se observan en las gráficas de las Fig. 3.3 y 3.4 respectivamente. Se puede apreciar que los picos más relevantes se hallan en los ejes X, Z.

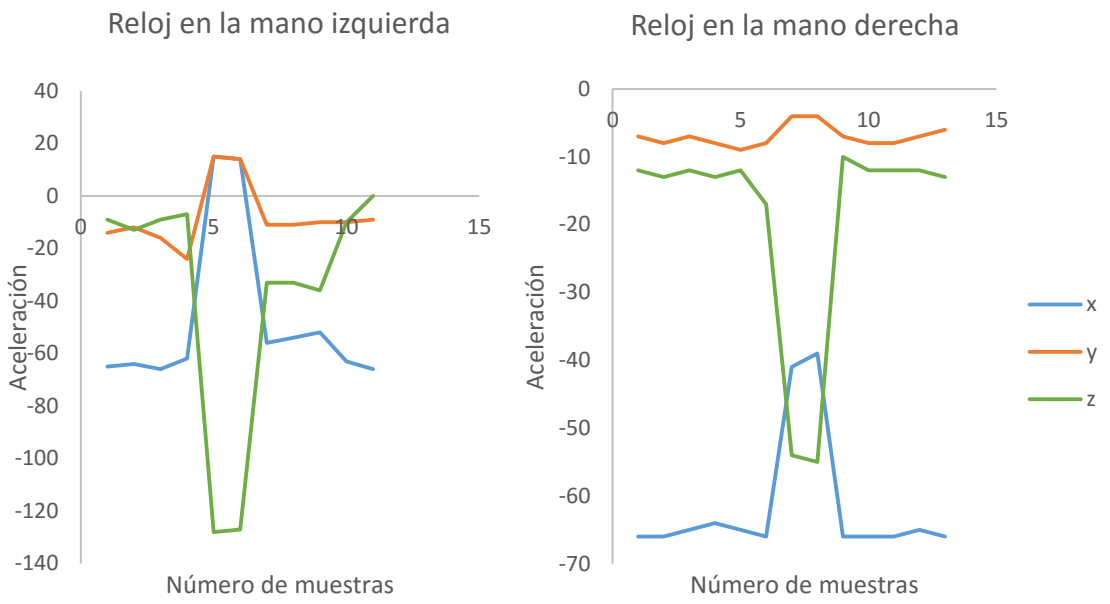

Fig. 3.3. Giro a la izquierda desde la posición de equilibrio 1

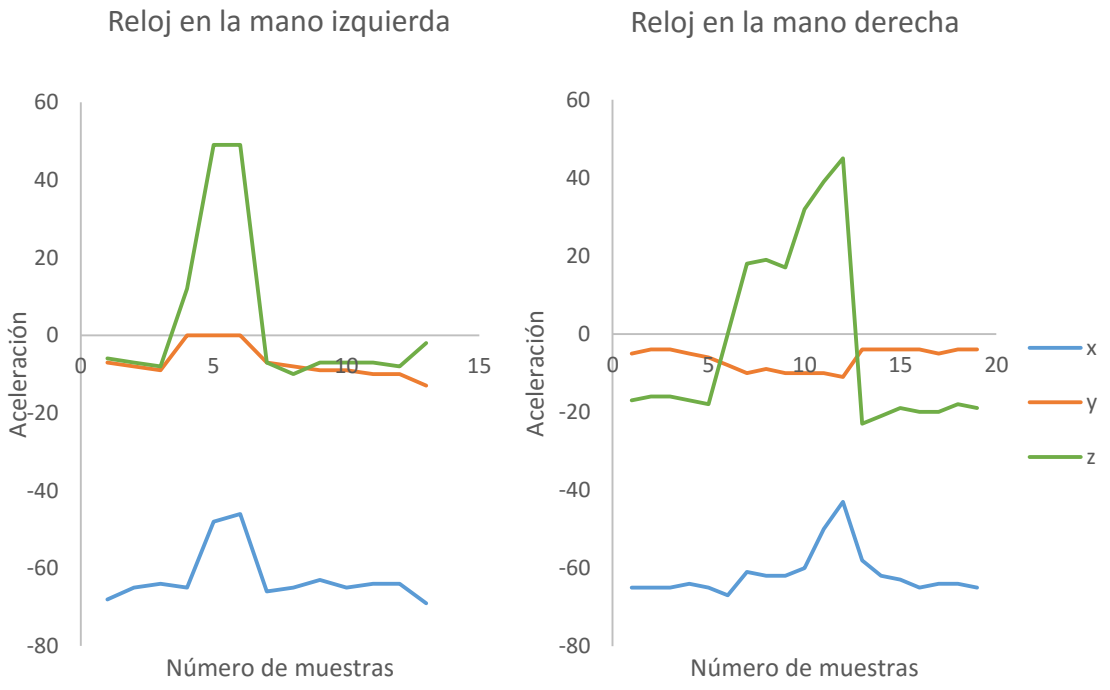

Fig. 3.4. Giro a la derecha desde la posición de equilibrio 1

Como en el caso anterior, tomando como origen la posición de equilibrio 1, flexionando el brazo arriba y abajo, resultan las gráficas de las Fig. 3.5 y 3.6 respectivamente. Esta vez, los ejes que proporcionan las muestras más significativas son X, Y.

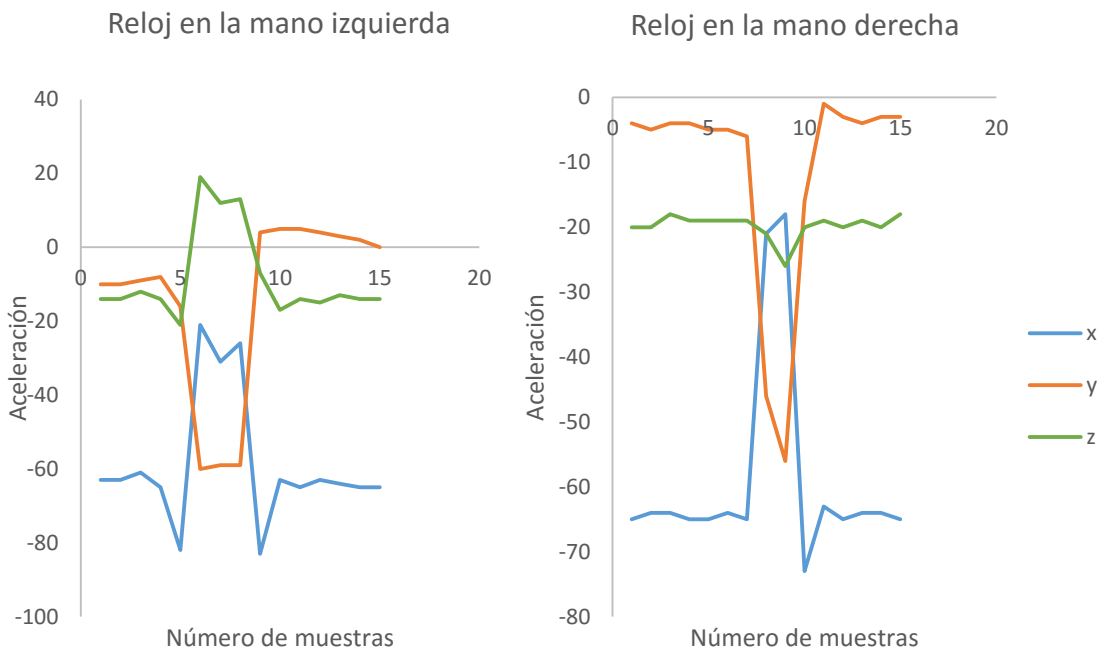

Fig. 3.5. Movimiento hacia arriba desde la posición de equilibrio 1

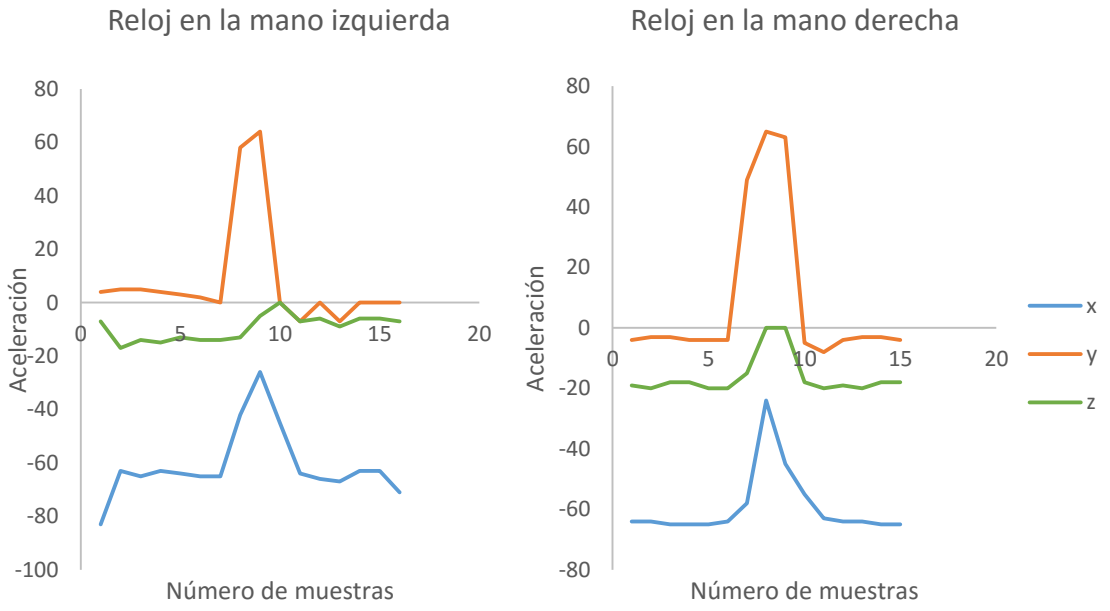

Fig. 3.6. Movimiento hacia abajo desde la posición de equilibrio 1

Finalmente, partiendo de la posición de equilibrio 2 y flexionando el brazo hacia arriba y abajo como en el caso precedente, se obtienen las curvas que pueden verse seguidamente en las Fig. 3.7 y 3.8. Los ejes de interés para la identificación de los patrones son Y, Z.

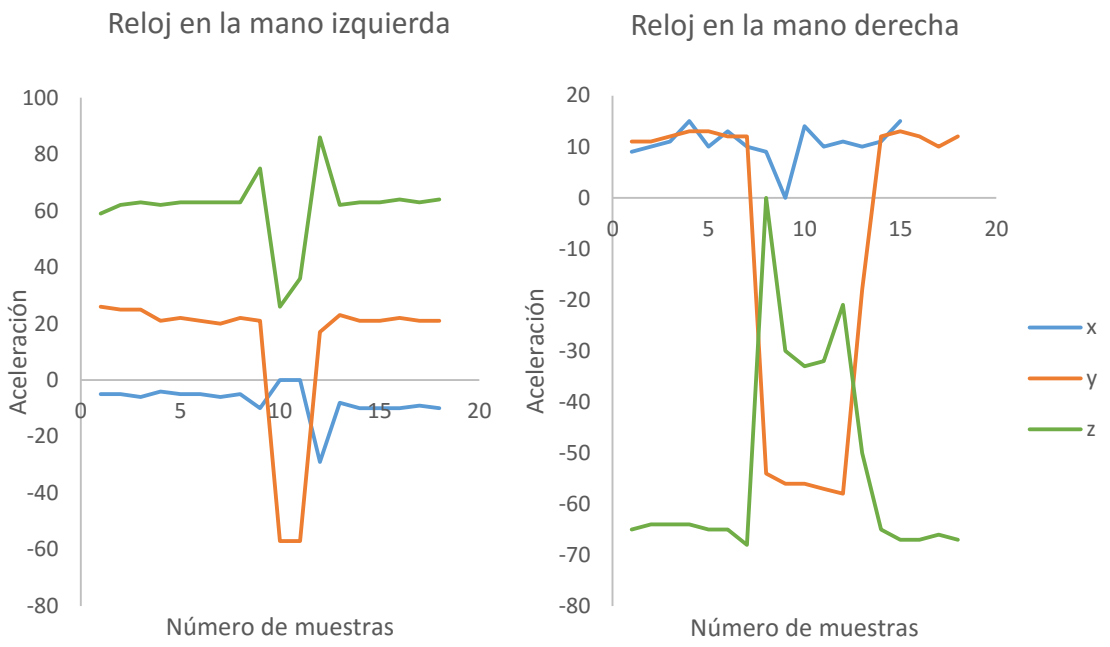

Fig. 3.7. Movimiento hacia arriba desde la posición de equilibrio 2

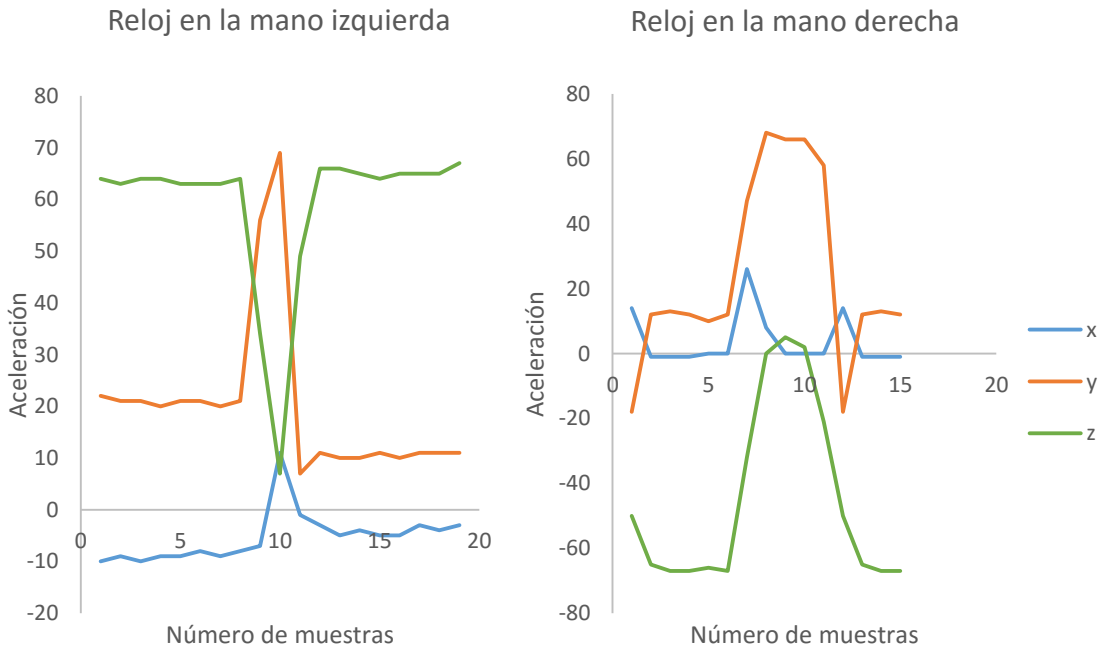

Fig. 3.8. Movimiento hacia abajo desde la posición de equilibrio 2

3.2. Modos de control

Se definieron dos modos de control, uno de cuatro y otro de seis grados de libertad. El primero permite mover el AR.Drone 2.0 hacia adelante, atrás, izquierda y derecha basándose en un único punto estable, la posición de equilibrio 1.

El segundo modo de control cuenta con de seis grados de libertad, gracias a que se basa en dos puntos de estabilidad, las posiciones de equilibrio 1 y 2. Dando así capacidad al dron para moverse en el plano longitudinal y también en el eje de la gravedad, es decir, hacia arriba y hacia abajo.

Tal y como ya se ha adelantado anteriormente, el movimiento elegido para enviar el comando de despegue fueron dos palmadas seguidas, siempre en menos de tres segundos. Por otro lado, en la lista que hay a continuación se adjunta cada comando de movimiento que puede enviarse al cuadricóptero, seguido del movimiento que hay que hacer con el brazo o muñeca. El envío de dichas órdenes supone un desplazamiento del dron de un metro aproximadamente.

- **Adelante.** Movimiento hacia arriba desde la posición de equilibrio 1
- **Atrás.** Movimiento hacia abajo desde la posición de equilibrio 1
- **Izquierda.** Giro a la izquierda desde la posición de equilibrio 1
- **Derecha.** Giro a la derecha desde la posición de equilibrio 1
- **Arriba.** Movimiento hacia arriba desde la posición de equilibrio 2
- **Abajo.** Movimiento hacia abajo desde la posición de equilibrio 2

Por último, para enviar el comando de aterrizaje, basta con efectuar dos movimientos hacia abajo desde la posición de equilibrio 2, es decir, mandar dos comandos de movimiento hacia abajo, siempre en menos de cinco segundos.

3.3. Detección de movimientos

Hecho el estudio de los movimientos y definidos los modos de control, se pasó a la detección de movimientos. Se probó con la técnica de aprendizaje automático (*machine learning*), y posteriormente se empleó un modelo basado en umbrales o valores mínimos.

3.3.1. *Machine learning*

El *machine learning* o aprendizaje automático, es el subcampo de la ciencia computacional y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan a las computadores aprender sin ser programados

explícitamente. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos [26].

Con el objetivo de conseguir una mayor adaptabilidad del sistema y generalizarlo para múltiples personas, se implementó *machine learning* para la detección del movimiento de despegue.

Más específicamente, se pensó en utilizar una técnica de aprendizaje automático supervisado. Este tipo de métodos se basan en un modelo predictivo. Según el conocimiento adquirido a partir de un conjunto de datos ya clasificados, el modelo asigna una etiqueta a cada elemento de entrada [27].

Con el objetivo de implementar *machine learning*, se creó un archivo CSV con los datos de aceleración de 200 movimientos de distintas personas, cada uno definido por 30 muestras, 10 por cada eje del eZ430-Chronos.

Del total de los 200 movimientos, 100 eran palmadas y los demás movimientos aleatorios. Es importante señalar que las palmadas se hicieron con el reloj colocado en la muñeca izquierda. A continuación, en las Fig. 3.9, 3.10 y 3.11 pueden verse los datos de las 100 palmadas que se incluyen en el archivo CSV, para realizar el aprendizaje supervisado.

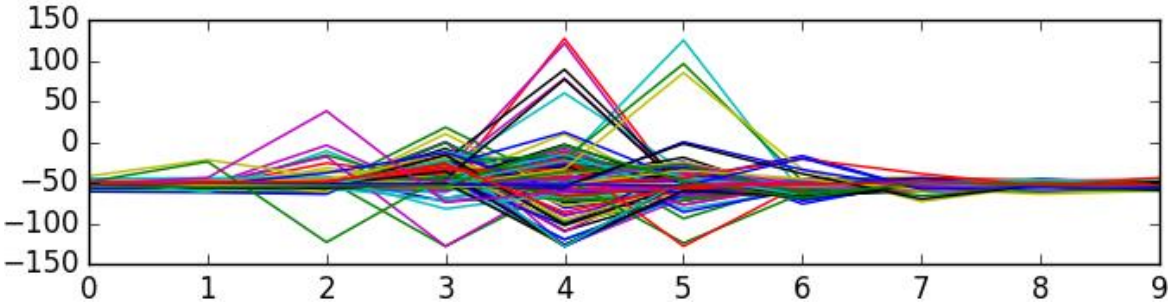

Fig. 3.9. Datos de aceleración de las palmadas en el eje X sin procesar

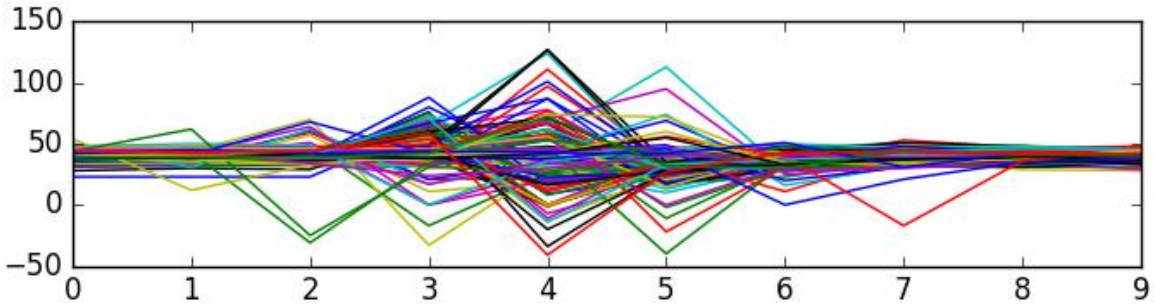

Fig. 3.10. Datos de aceleración de las palmadas en el eje Y sin procesar

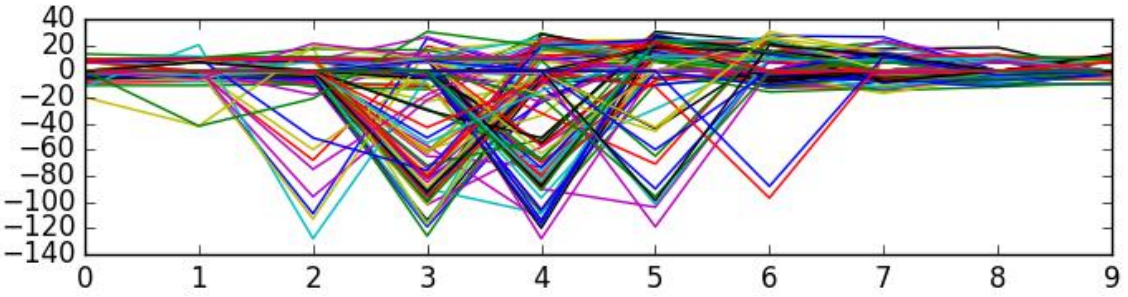

Fig. 3.11. Datos de aceleración de las palmadas en el eje Z sin procesar

Como puede observarse, las palmadas no se efectuaron exactamente en el mismo instante, ya que los picos aparecen en varias muestras distintas en lugar de siempre en la misma. Para mejorar el rendimiento de las predicciones de *machine learning*, se procesaron los datos muy superficialmente, simplemente se centraron los picos de las aceleraciones en el eje Z justo en la mitad de las 10 muestras.

En las siguientes Fig. 3.12, 3.13 y 3.14 se encuentran las gráficas resultantes con los datos procesados.

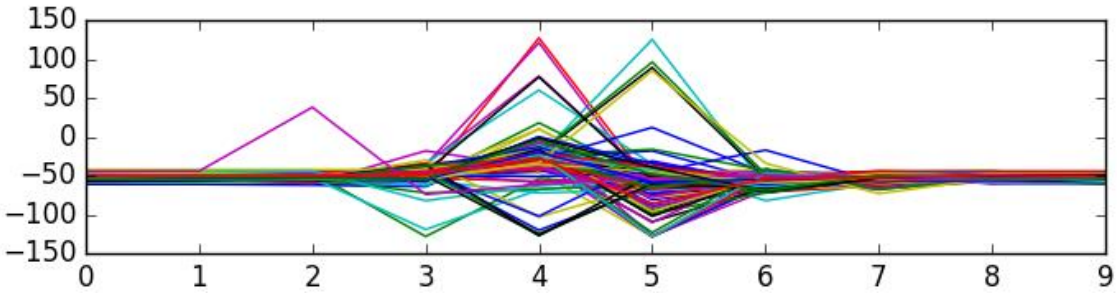

Fig. 3.12. Datos de aceleración de las palmadas en el eje X procesados

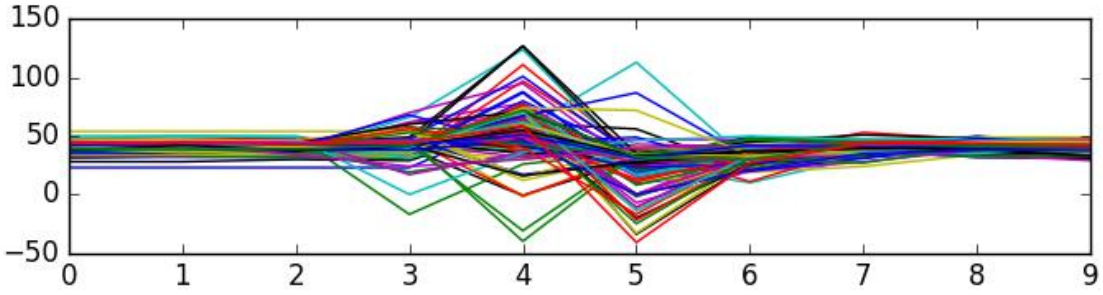

Fig. 3.13. Datos de aceleración de las palmadas en el eje Y procesados

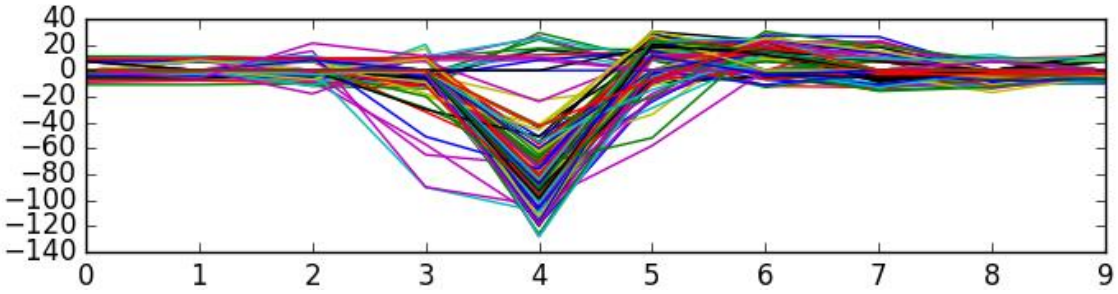

Fig. 3.14. Datos de aceleración de las palmadas en el eje Z procesados

Gracias al módulo *sklearn* de Python [28], se probaron diversos algoritmos de clasificación. Se eligió el árbol de decisión ya que era el algoritmo que requería menos preprocesado de los datos, y dio mejores resultados que otros que se probaron.

El árbol de decisión, es un algoritmo clasificador que consta de hojas, nodos y ramas. Las hojas no son más que nodos de los cuales no surge ninguna rama, como puede verse en la Fig. 3.15.

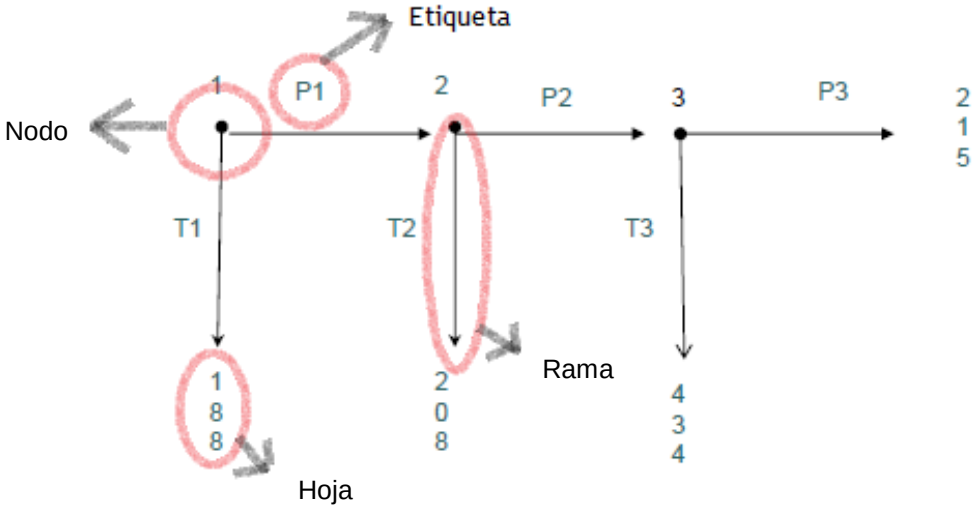

Fig. 3.15. Esquema de la estructura de un árbol de decisión

De cada nodo nacen dos o más ramas en las que se formulan proposiciones lógicas excluyentes, las cuales dividen el espacio de atributos con distintas etiquetas. Esto quiere decir que cada instancia, dependiendo de los valores de los atributos implicados en la proposición lógica de un nodo, cumple la condición correspondiente a una de las ramas.

De esta manera, comenzando en el nodo raíz, cada instancia se dirige hasta una de las hojas, y a todas las instancias pertenecientes a la misma hoja se les asigna una etiqueta común [29].

En general la técnica de *machine learning* no dio buenos resultados, su tasa de error era demasiado grande, ya que en ocasiones se detectaban falsos positivos, o se hacían palmadas y no eran detectadas. Dada la gran no-linealidad del movimiento humano, las muestras a predecir eran muy cambiantes, con lo cual era necesario hacer un trabajo de preprocesado mucho más profundo, para así poder normalizar más todavía las muestras.

Sin embargo, el preprocesado del que se habla no llegó a implementarse, puesto que tras múltiples pruebas, se observó que los métodos y algoritmos de *sklearn* ralentizaban el proceso de identificación de los movimientos. Consecuentemente el sistema entero tenía un retraso de entorno a los 2 segundos, que no era conveniente asumir.

Además, debido a la alta no-linealidad de las muestras, las predicciones no eran del todo fiables, siendo el porcentaje de error de detección demasiado a alto como para poder pilotar un dron. Con lo cual, se pasó a otro método de detección de movimientos más simple basado en la definición de umbrales.

En el Anexo V se encuentra un programa escrito en Python que lee los datos del reloj eZ430-Chronos, y aplica *machine learning* para detectar si se ha producido una palmada.

3.3.2. Umbrales

Tras probar la técnica de aprendizaje supervisado y ver la gran tasa de error que comportaba, se cambió de método de detección de movimientos.

Anteriormente, al hacer el estudio de los patrones de los movimientos de interés (ver 3.1. Estudio de los movimientos), se vieron las grandes diferencias entre ellos. Los ejes donde era más evidente el movimiento, es decir, donde había una diferencia de valores más significativa, cambiaba para cada uno de ellos, así como los valores máximos y mínimos.

Así pues, se asignaron dos valores mínimos o umbrales a cada movimiento, uno referente a la mano izquierda y otro a la derecha, con el objetivo de poder identificarlos. Para cada persona el valor de los umbrales puede cambiar, con lo cual para generalizar el sistema sería necesario hacer una calibración previamente.

Se definió la detección del movimiento de interés, cuando el valor obtenido a partir de los acelerómetros del reloj excediera el umbral. La probabilidad de error es bastante baja, ya que los picos de los movimientos hacia la izquierda y derecha tienen signos contrarios en el eje Z, mientras que los movimientos hacia delante y atrás tienen picos de signo opuesto en el eje Y, justo igual que en el caso de los movimientos hacia arriba y abajo.

El diagrama de flujo de la Fig. 3.16 muestra la secuencia que se sigue desde que se adquieren los datos del reloj, hasta que se realiza el envío del comando correspondiente al dron.

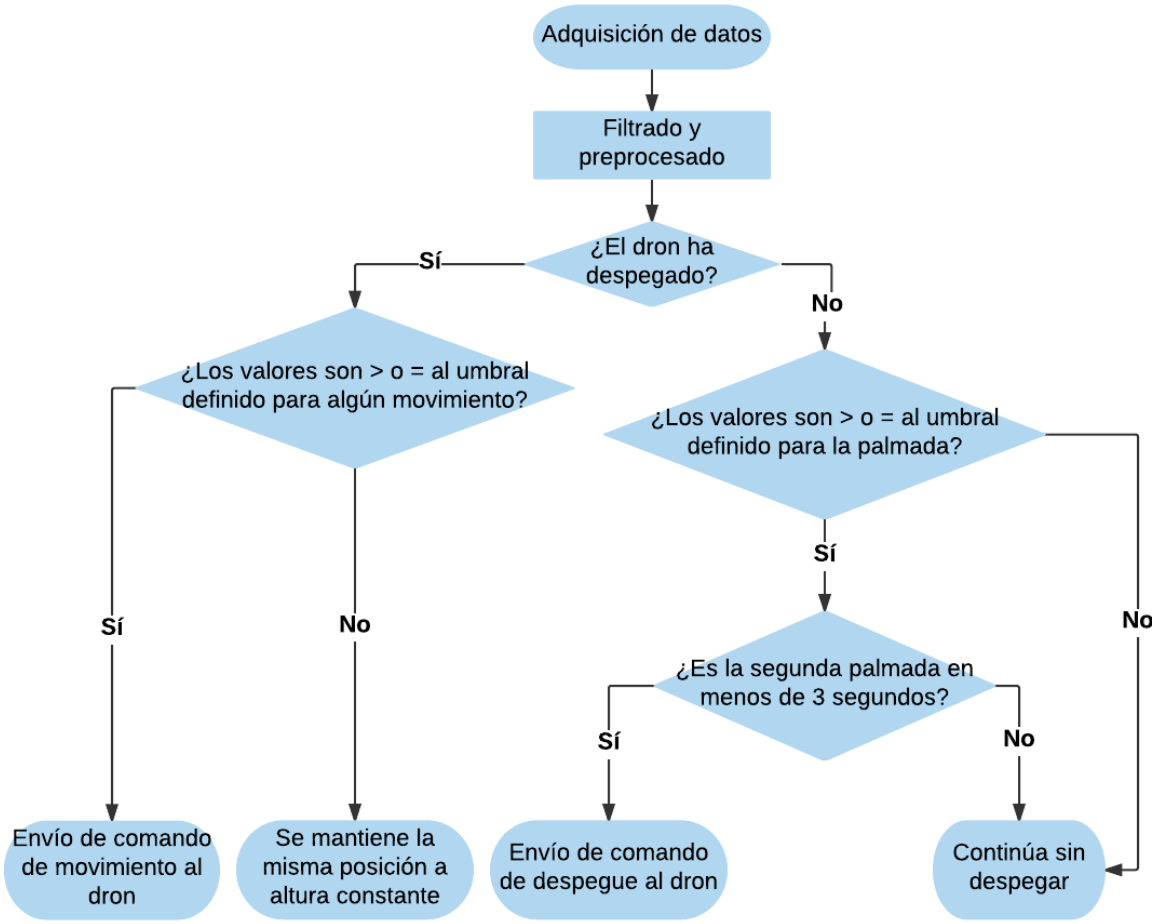

Fig. 3.16. Diagrama de flujo de la detección de movimientos y envío de comandos mediante umbrales

El primer paso es realizar la adquisición de datos de aceleración del reloj eZ430-Chronos, una vez se obtienen se debe aplicar un preprocesado y filtrado (ver 1.3. Diseño e implementación del sistema de adquisición).

Una vez se han tratado los datos, a partir de los datos de navegación del AR.Drone 2.0, se comprueba que haya despegado. En caso afirmativo, se comprueba si los valores son iguales o superiores al umbral definido para algún movimiento, siempre teniendo en cuenta la posición de equilibrio de la cual parte la mano del usuario. Si los valores exceden el umbral, se envía el comando de movimiento al dron con la orden correspondiente (izquierda, derecha, delante, atrás, arriba o abajo). En cambio, si no se excede el umbral, envía el comando de *hover*, de forma que el cuadricóptero se mantiene en la misma posición longitudinal y la misma altura.

Si por el contrario, el AR.Drone 2.0 se encuentra en el suelo, es decir, que aún no ha despegado, se comprueba si los valores de aceleración leídos son iguales o superiores a los umbrales definidos para una palmada. En caso negativo el dron continúa en el suelo sin despegar. Por otra parte, si los valores son efectivamente superiores, y es la segunda vez que se detectan datos que

sobrepasan el umbral, es decir, es la segunda palmada detectada, se realiza el envío del comando de despegue al dron. Si es la primera palmada que se detecta, el dron sigue en el suelo sin despegar.

Todo este proceso explicado, se realiza para cada paquete de datos de aceleración leído. Una vez se acaba la secuencia, se vuelve a repetir todo el proceso para el siguiente el siguiente paquete de valores de aceleración.

La comparación de los valores leídos del reloj con el umbral supone mucho menos tiempo de cálculo que los algoritmos usados en *machine learning*, por lo que el tiempo de detección en relación a dicho método, disminuyó considerablemente.

CAPÍTULO 4. INTEGRACIÓN DE SISTEMAS Y RESULTADOS

En este capítulo se explica cómo se ha llevado a cabo la integración de los sistemas ya presentados en secciones anteriores, por una parte la adquisición de movimientos y por otra el vehículo aéreo no tripulado, se comentarán también los resultados obtenidos.

4.1. Integración de sistemas

Para la integración de ambos sistemas, se combinaron los códigos en Python que se habían hecho para obtener los datos del reloj eZ430-Chronos y para controlar el AR.Drone 2.0. La estructura del programa final es la que aparece en el diagrama de la Fig. 4.1.

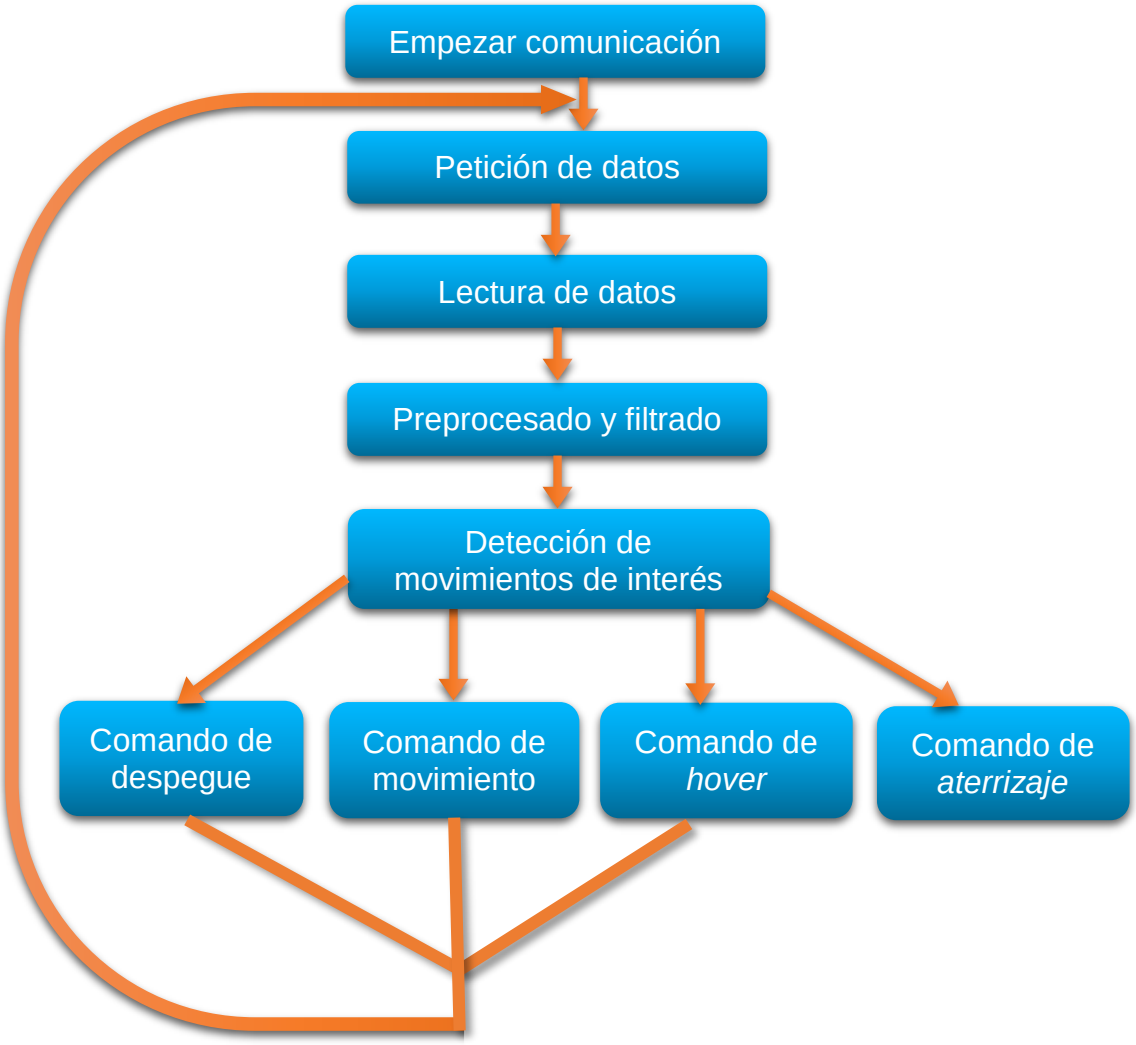

Fig. 4.1. Diagrama secuencial de funciones del programa final

Tal y como puede observarse en la Fig. 4.1, la primera función que realiza el programa es establecer las conexiones y empezar las comunicaciones, no solo con el reloj sino que también con el dron, ya que es importante verificar que no se encuentra en modo de emergencia (ver 2.2.2 Modelo de comunicaciones).

Una vez abiertas las comunicaciones y comprobado el estado del AR.Drone 2.0, se envía un paquete de petición de datos al eZ430-Chronos, seguidamente cuando se obtiene el paquete de respuesta por parte del reloj se hace la lectura del mismo. Luego, se realiza un preprocesado y filtrado de los datos proporcionados por el eZ430-Chronos (ver 1.3. Diseño e implementación del sistema de adquisición).

Después de que los datos de aceleración hayan sido debidamente tratados, se procede a la detección de movimientos de interés, mediante el método de umbrales (ver 3.3.2. Umbrales).

En primer lugar, cuando el dron aún está en el suelo, se busca detectar el movimiento de despegue, los demás se ignoran. Cuando el dron ya ha despegado, el movimiento de despegue no se vuelve a tener en cuenta. Una vez ya se ha enviado el comando despegue, si el algoritmo de detección no capta ningún movimiento, se envía al dron el comando de *hover* para que se quede en la misma posición a altitud constante. En caso de que sí se detecte algún movimiento, se procede a hacer el envío del comando correspondiente a la actividad detectada.

Una vez se realiza el envío del comando correspondiente, se vuelve a comenzar todo el proceso de nuevo, enviando otro paquete de petición de datos al reloj. El mismo procedimiento se repite hasta que se detecta el movimiento de aterrizaje.

4.2. Ensayos y resultados

Con la integración de sistemas completamente terminada, se pasó a la parte práctica. Se realizaron múltiples pruebas de vuelo, esta vez ya con la arquitectura completa del sistema, para comprobar su correcto funcionamiento.

Los vuelos realizados fueron de unos 4 minutos de media cada uno, se probaron los dos modos de control definidos, tanto el de cuatro, como el de seis grados de libertad. Además, se probó el pilotaje con el reloj eZ430-Chronos en la mano izquierda y en la derecha, para ver si había algún tipo de cambio en referencia a la maniobrabilidad del dron.

Cabe destacar que todos los ensayos se realizaron en interiores como puede verse en la siguiente Fig. 4.2, para que no hubiera ningún tipo de perturbación a causa del viento. También puede observarse, que se colocó una imagen asimétrica y de alto contraste en el suelo.

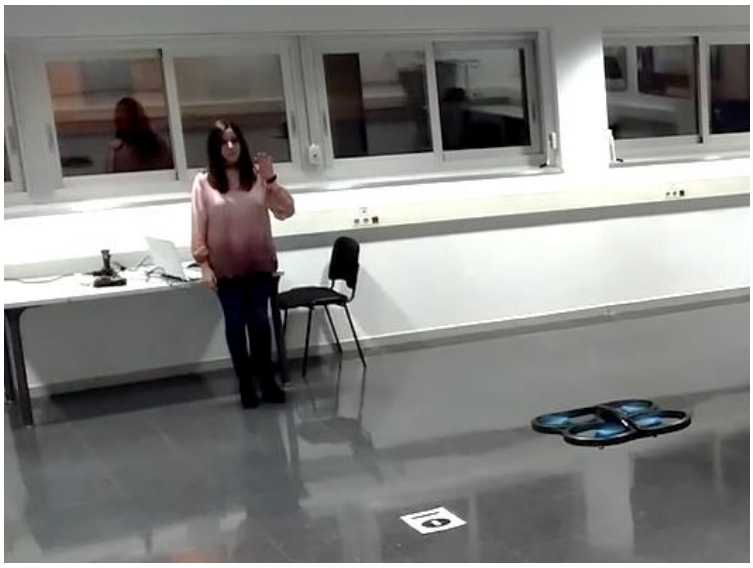

Fig. 4.2. Prueba de vuelo

Teniendo en cuenta, que los giros de pequeño rango no se contemplan en la modelización de los movimientos, ya que se basa en movimientos amplios, se añadieron ayudas sonoras. Pudiendo así orientar y hacer saber al piloto que el movimiento efectuado era lo suficientemente amplio como para ser captado por el programa.

Estas referencias sonoras fueron especialmente útiles para controlar mejor la trayectoria, ya que si después de realizar un movimiento, el usuario tarda mucho en volver poner la mano en posición de equilibrio, se corre el riesgo de que el comando correspondiente se envíe dos veces en lugar de una, siendo entonces necesario corregir la maniobra.

También se tuvieron que ajustar parámetros que concernían directamente a la adquisición de movimientos y al vuelo:

- **Se disminuyó el tiempo de desplazamiento del dron.**
  Originalmente, cualquier desplazamiento tanto en el plano longitudinal como en el eje vertical, se desarrollaba durante un tiempo de era de 2 segundos 0.5 segundos

- **Se incrementó la velocidad de desplazamiento del dron.**
  La velocidad de movimiento pasó a ser de un 10% a un 30% en el plano longitudinal. Por otro lado, en el caso del movimiento ascendente y descendiente, se cambió de 30% a 40% de velocidad.

Todas estas modificaciones se hicieron para mejorar el tiempo de respuesta, pues en los primeros vuelos el sistema era demasiado lento como para poder integrarlo en ámbitos artísticos tales como la danza, lo cual fue la motivación de este proyecto.

Finalmente, tras varios ensayos, se consiguió un sistema de control de rápida respuesta, a su vez con gran estabilidad. Pero tal y como sucede con cualquier tipo de dispositivo de pilotaje, es importante destacar que es necesario que el usuario tenga una cierta práctica para controlar el vuelo a la perfección. En el caso de este sistema, el aprendizaje por parte del piloto es bastante rápido ya que los controles son muy intuitivos.

CAPÍTULO 5. CONCLUSIONES Y TRABAJO FUTURO

En este último capítulo se exponen todas las conclusiones sacadas de la elaboración de este trabajo de fin de grado. También se analizan las perspectivas y el posible trabajo futuro que podría hacerse a partir de este proyecto.

5.1. Conclusiones

Tras la finalización del proyecto, se han podido concluir ciertos aspectos relacionados, con los dispositivos empleados, la modelización y detección de los movimientos y la integración de sistemas.

La plataforma usada para formar el subsistema de adquisición de movimientos, ha resultado ser muy adecuada. El eZ430-Chronos de Texas Instruments, es un dispositivo inalámbrico de pequeño tamaño, por lo que no limita ni interfiere en los movimientos del usuario. Además, la transferencia de los datos del reloj al RF Access Point conectado al PC, es lo suficientemente rápida para el objetivo del proyecto, pudiendo obtener un paquete de datos cada 45 ms en el mejor de los casos.

Por el contrario, en referencia al vehículo aéreo no tripulado, después de numerosas pruebas de vuelo, se ha visto que solo uno de los dos cuadricópteros estudiados podía utilizarse para el fin de este proyecto.

El Crazyflie 2.0 de Bitcraze es un dispositivo interesante, dado que es un instrumento pensado específicamente para la investigación y desarrollo, además, su tamaño es ideal para esta aplicación. Sin embargo por sí solo no es capaz de mantener un vuelo estable. La falta de un sistema de posicionamiento, hace que su vuelo sea tremendamente inestable y en ocasiones incontrolable. A partir de los resultados obtenidos en las pruebas de vuelo, se concluyó que no era adecuado para el objetivo que se perseguía.

Por otro lado, el AR.Drone 2.0 de Parrot, sí que puede considerarse como un dispositivo válido, ya que su vuelo es muy estable, y con un gran margen de maniobrabilidad. No obstante, tiene sus desventajas respecto al Crazyflie 2.0, puesto que su tamaño es mucho mayor, y puede ser resultar excesivo para realizar vuelos en interiores.

En relación a la modelización de movimientos, se determinó que es imprescindible hacer un estudio detallado de cada posición y movimiento de interés, dado que el movimiento de una persona es altamente no-lineal. Es igualmente importante considerar si el reloj se coloca en la mano izquierda, o en la derecha. Los patrones para un mismo movimiento difieren en función de la ubicación del reloj.

Considerando el estudio de los movimientos, se definieron dos modos de control uno de 4 grados de libertad (izquierda-derecha-adelante-atrás) y otro de 6 (izquierda-derecha-adelante-atrás-arriba-abajo).

Dichos modos en la práctica funcionaron correctamente y comparado con los medios convencionales, simplificaban enormemente el pilotaje.

La detección de movimientos es quizás, la parte más crítica de todo el proyecto, ya que el algoritmo tiene que ser rápido y debe tener una probabilidad de error baja, es por eso que se probaron dos métodos de detección diferentes basados en: *machine learning* y umbrales.

La detección mediante *machine learning*, no fue efectiva. Pese a que los datos de aprendizaje, se procesaron para obtener una mejor respuesta, el error era muy elevado. Se detectaron multitud de falsos positivos y además tenía un tiempo de ejecución demasiado grande, con lo que generaba un retraso no asumible para el sistema de control de un dron, por lo que acabó por descartarse.

No obstante, con el método de detección basado en umbrales, se consiguió una respuesta mucho mejor. En términos de error, algunos movimientos pueden no ser detectados si no son lo suficientemente amplios. En cuanto al tiempo de ejecución, este algoritmo es mucho más rápido, de modo que no se genera ningún retraso importante. Aunque está técnica tiene un buen rendimiento, para movimientos más complejos puede ser insuficiente.

En referencia a la integración de sistemas, se ha visto que las ayudas sonoras son muy útiles para controlar mejor la trayectoria, ya que se reduce la corrección de maniobras.

Con este proyecto se quería crear un sistema de control que pudiera ser utilizado en espectáculos o ámbitos artísticos, como la danza. Se puede concluir que el sistema es apto en función del tipo de efecto o respuesta que se quiera conseguir. Si se busca un control muy rápido, no es conveniente usar el sistema de control que se ha definido, ya que los movimientos del usuario deben ser muy precisos y controlados. Si por el contrario se busca un control más lento, resulta muy adecuado.

5.2. Trabajo futuro

Como ya se ha comentado, debido a la inestabilidad de su vuelo, se tuvo que descartar la utilización del Crazyflie 2.0. Sin embargo existe un sistema de posicionamiento local diseñado por Bitcraze, que otorga más precisión y estabilidad.

Se basa en el cálculo de distancias relativas a varios receptores colocados en diferentes puntos del espacio donde va a tener lugar el vuelo, y de los cuales se sabe su posición exacta. En la Fig. 5.1 se puede ver un esquema con la idea del principio de funcionamiento del sistema, en dos dimensiones. No obstante, dicho sistema se encuentra aún en fase de pruebas.

En Fig. 5.2 se puede ver la instalación necesaria para la utilización del sistema.

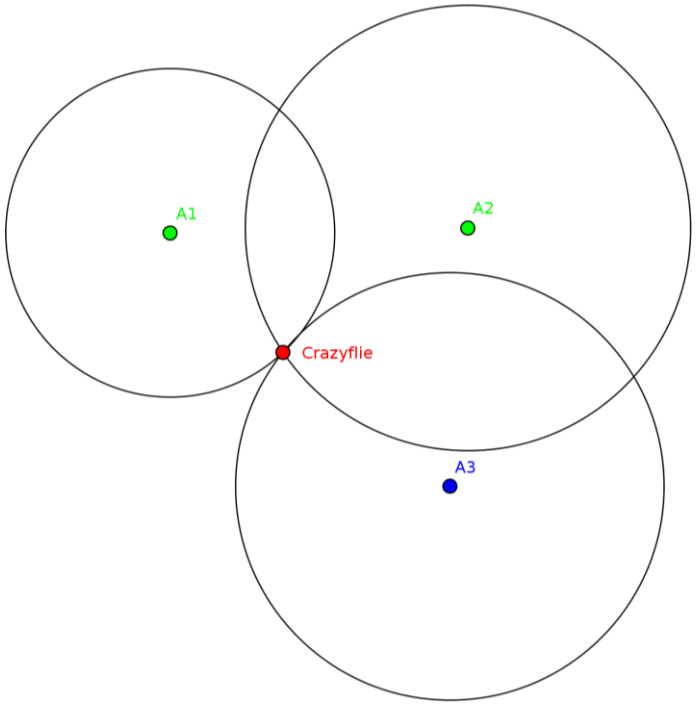

Fig. 5.1. Esquema del principio de funcionamiento del *Loco Positioning System* de Bitcraze

El funcionamiento es simple, una vez se activa algún el comando que se quiere enviar, el PC con ROS envía la nueva posición del dron mediante la Crazyradio PA. Seguidamente el Crazyflie 2.0 calcula su posición actual así como la nueva, tomando los nodos (*anchors*) como referencia. Una vez hecho el cálculo, el dron se dirige a la nueva posición o *setpoint*.

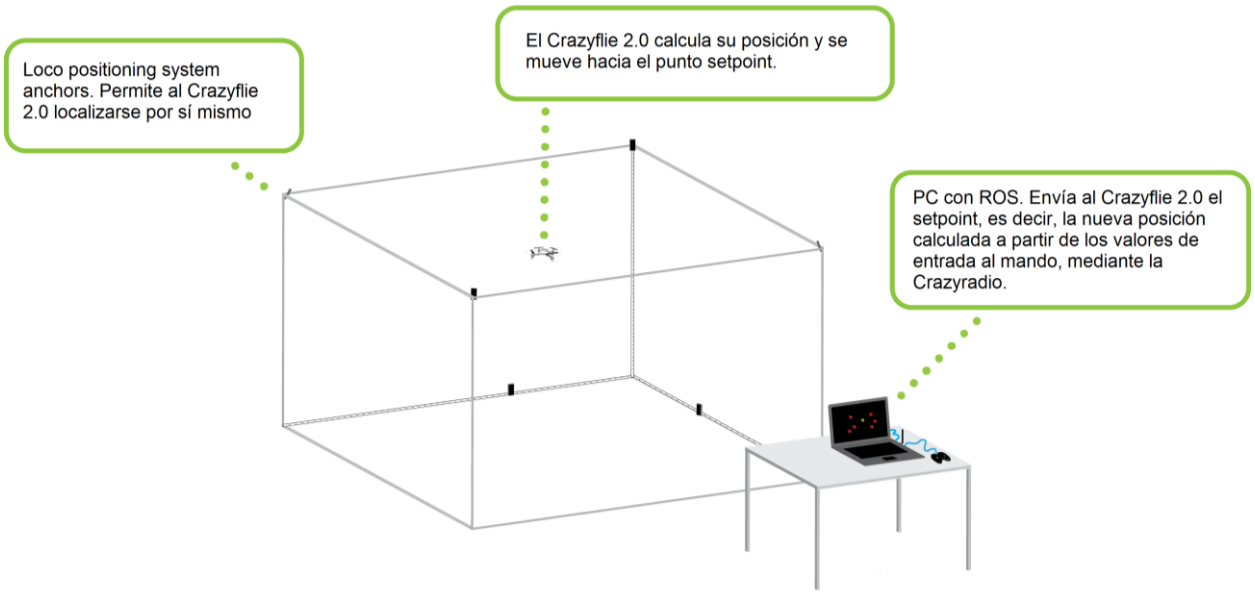

Fig. 5.2. Instalación del *Loco Positioning System*

En cuanto a la fase más crítica del sistema, es decir, la detección de movimientos, la técnica de aprendizaje automático supervisado o *machine learning* puede llegar a ser implementada.

Se podría tratar los datos de manera mucho más profunda, normalizando valores, indicando el inicio y fin de la actividad o buscando relaciones entre los distintos ejes. Todo este procesado previo, ayudaría a obtener una mejor predicción del algoritmo clasificador.

REFERENCIAS

[1] European Comission. (Abril 2014). *European Commission calls for tough standards to regulate civil drones* <http://europa.eu/rapid/press-release_IP-14-384_en.htm>

[2] The Washington Post. Fung, Brian. (Agosto 2013). *Why drone makers have declared war on the word 'drone'* <https://www.washingtonpost.com/news/the-switch/wp/2013/08/16/why-drone-makers-have-declared-war-on-the-word-drone/?utm_term=.b0451bb7bc49>

[3] Small Drone Review. (Octubre 2015). *Best Farming Drones – UAV for Agriculture* <http://smalldronesreview.com/2015/10/07/best-farming-drones-uav-for-agriculture/>

[4] CBS News. (Diciembre 2013). *DHL testing delivery drones* <http://www.cbsnews.com/news/dhl-testing-delivery-drones/>

[5] Bloomberg Technology. Moskwa, Wojciech. (Mayo 2016). *World Drone Market Seen Nearing $127 Billion in 2020, PwC Says* <https://www.bloomberg.com/news/articles/2016-05-09/world-drone-market-seen-nearing-127-billion-in-2020-pwc-says>

[6] Massachusetts Institute of Technology. Barkley Graham, Brian. (Mayo 2000). *Using an Accelerometer Sensor to Measure Human Hand Motion* <http://www-mtl.mit.edu/researchgroups/MEngTP/Graham_Thesis.pdf>

[7] Parrot. *Parrot for developers* <http://developer.parrot.com/products.html>

[8] Jet Brains. *Pycharm* <https://www.jetbrains.com/pycharm/>

[9] Texas Instruments. *EZ430 Chronos* <http://processors.wiki.ti.com/index.php/EZ430-Chronos>

[10] Texas Instruments. (Julio 2010). *Chronos High 5 Me!* <http://processors.wiki.ti.com/index.php/Chronos_High_5_Me!>

[11] National Instruments. *Software de Desarrollo de Sistemas NI LabVIEW* <http://www.ni.com/labview/esa/>

[12] Python. *Python* <https://www.python.org/>

[13] Python. (Agosto 2013). *PEP 8 -- Style Guide for Python Code* <https://www.python.org/dev/peps/pep-0008/>

[14] Bitcraze. Crazyflie 2.0 <https://www.bitcraze.io/crazyflie-2/>

[15] Bitcraze. (Febrero 2016). *Crazyflie 2.0* <https://wiki.bitcraze.io/projects:crazyflie2:index>

[16] Google Play. (Abril 2016). *Crazyflie Client* <https://play.google.com/store/apps/details?id=se.bitcraze.crazyfliecontrol2>

[17] iTunes. (Septiembre 2015). *Crazyflie 2.0* <https://play.google.com/store/apps/details?id=se.bitcraze.crazyfliecontrol2>

[18] Bitcraze. (Abril 2016). *Crazyflie PC client 2016.4* <https://github.com/bitcraze/crazyflie-clients-python/releases>

[19] Syracuse University. Martin, Catherine. (Mayo 2015). *Motion in the Field: A Study of Movement in Computer Science* <http://surface.syr.edu/cgi/viewcontent.cgi?article=1910&context=honors_capstone>

[20] Bitcraze wiki. (Julio 2015). *This is the analyses of finding a PWM to thrust transfer function* <https://wiki.bitcraze.io/misc:investigations:thrust>

[21] Parrot. *Parrot AR.Drone 2.0 Power Edition* <https://www.parrot.com/es/drones/parrot-ardrone-20-power-%C3%A9dition#ar-drone-20-power-edition>

[22] Google Play. (Agosto 2016). *AR.FreeFlight 2.4.15* <https://play.google.com/store/apps/details?id=se.bitcraze.crazyfliecontrol2>

[23] iTunes. (Octubre 2015). *FreeFlight2.0* <https://itunes.apple.com/es/app/freeflight/id373065271?mt=8>

[24] Mellado Bataller, Ignacio. *Sistema de control visual para micro-vehículos aéreos* <http://www.ignaciomellado.es/img/IgnacioMelladoBataller-PFM.pdf>

[25] afg984. (2015). *Welcome to pyardrone's documentation!* <http://pyardrone.readthedocs.io/en/latest/>

[26] Samuel, Arthur L. (1959). *"Some studies in machine learning using the game of checkers".* IBM Journal of research and development.

[27] Shai Shalev-Shwartz; Shai Ben-David. (2014). *Understanding Machine Learning: From Theory to Algorithms* <http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf>

[28] Scikit learn. *Scikit learn* <http://scikit-learn.org/stable/index.html>

[29] Universidad Autónoma de Madrid. Di Deco Sampedro, Javier. (Junio 2012). *Estudio y aplicación de técnicas de aprendizaje automático orientadas al ámbito médico: estimación y explicación de predicciones individuales* <https://repositorio.uam.es/bitstream/handle/10486/12100/59264_Di_Deco_Sampedro_JavierPFM.pdf?sequence=1>

ANEXO I. PROGRAMA DE ADQUISICIÓN DE DATOS EN C#

En este anexo, se incluye el código en C# que se hizo para verificar cuales eran los mensajes o paquetes a enviar desde el PC, para comunicarse con el reloj eZ430-Chronos.

El programa en primer lugar envía el paquete para iniciar la comunicación con el eZ430-Chronos. Después envía paquetes de petición de datos, lee y muestra en consola los datos que el reloj envía junto con el número de paquete, durante el tiempo en segundos que se indique en la variable `max_time`.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;
using System.Threading;
using System.IO;

namespace ReadSerialPorteZ430Chronos
{
    class MainClass
    {
        public static void Main (string[] args)
        {
            SerialPort myPort = new SerialPort ("COM4", 115200, Parity.None,
             8, StopBits.One);

            myPort.Open();

            byte[] cmdByte = new byte[3];

            // Start message:
            cmdByte [0] = 255;
            cmdByte [1] = 7;
            cmdByte [2] = 3;

            myPort.Write (cmdByte, 0, 3);

            double initial_time = DateTime.Now.Ticks/TimeSpan.TicksPerSecond;
            int sample_time = 45;
            double max_time = 60;
            string inbyteToString;

            int i = 0;

            while (DateTime.Now.Ticks/TimeSpan.TicksPerSecond
             initial_time <= max_time){
                if (myPort.BytesToRead > 0) {
                    byte[] inbyte = new byte[myPort.BytesToRead];
                    myPort.Read (inbyte, 0, inbyte.Length);

                    inbyteToString = BitConverter.ToString(inbyte).Replace ("
```

```
                    -", " ");
                    Console.WriteLine (inbyteToString + " " + "Package number
                    : " + i.ToString());

                    i++;
                }
                cmdByte = new byte[7];

                // Request message:
                cmdByte [0] = 255;
                cmdByte [1] = 8;
                cmdByte [2] = 7;

                myPort.Write (cmdByte, 0, 7);

                Thread.Sleep (sample_time);
            }

            myPort.Close ();

            Console.WriteLine("Press any key to continue...");
            Console.WriteLine();
            Console.ReadKey();
            myPort.Close();
        }
    }
}
```

ANEXO II. ALGORITMOS DE PREPROCESADO Y FILTRADO

En este anexo se encuentran los algoritmos utilizados para el preprocesado y filtrado de valores de aceleración del reloj eZ430-Chronos, tanto los del programa en LabVIEW como en Python.

Python

Este primer método, elimina el ruido ignorando los cambios aceleración muy pequeños. Además, determina en qué dirección se produce la dirección, en caso necesario escala el valor de aceleración y le asigna un signo negativo.

```
def filter_acceleration(self, x_axis_acceleration, x_samples_counter):
    if x_axis_acceleration[x_samples_counter] <= 6:  # threshold
        x_axis_acceleration[x_samples_counter] = 0
    elif x_axis_acceleration[x_samples_counter]-
    x_axis_acceleration[x_samples_counter-1] <= 6:
        x_axis_acceleration[x_samples_counter] =
               x_axis_acceleration[x_samples_counter]

    if x_axis_acceleration[x_samples_counter] > 127:  # negative
    direction
        x_axis_acceleration[x_samples_counter] = (255 -
               x_axis_acceleration[x_samples_counter] + 1) * -1
```

La siguiente función, actúa como un filtro paso alto, se queda con las aceleraciones mayores a un umbral elevado. Esto se traduce físicamente en tener en cuenta únicamente los movimientos de amplio rango.

```
def rough_movement_limitation(self, x_acc, y_acc, samples_counter):
    xnew, ynew = self.single_axis_limitation(x_acc, y_acc,
                  samples_counter)

    if xnew < -30:
        xnew = -127
    elif xnew > 30:
        xnew = 127
    else:
        xnew = 0

    if ynew < -30:
        ynew = -127
    elif ynew > 30:
        ynew = 127
    else:
        ynew = 0
    return xnew, ynew,
```

Por último, el siguiente método comprueba en cuál de los dos ejes de entrada se tiene un valor de aceleración mayor. El eje donde se tiene el mayor valor no se modifica, mientras que el valor del otro eje se iguala a 0, de modo que a efectos prácticos se tienen en cuenta los movimientos en un único eje.

```
def single_axis_limitation(self, x_axis_acceleration,
y_axis_acceleration, watch_samples_counter):
    if abs(x_axis_acceleration[watch_samples_counter]) >
        abs(y_axis_acceleration[watch_samples_counter]) and
        abs(x_axis_acceleration[watch_samples_counter-1]) >
        abs(y_axis_acceleration[watch_samples_counter-1]):
         ynew = 0
         return x_axis_acceleration[watch_samples_counter], ynew
    else:
         xnew = 0
         return xnew, y_axis_acceleration[watch_samples_counter]
```

LabVIEW

El siguiente código de LabVIEW muestra la implementación del modo de filtrado que limita a un solo eje. Como en el caso de Python se comprueba en qué eje se tiene una mayor aceleración, y el valor del otro eje se iguala a 0.

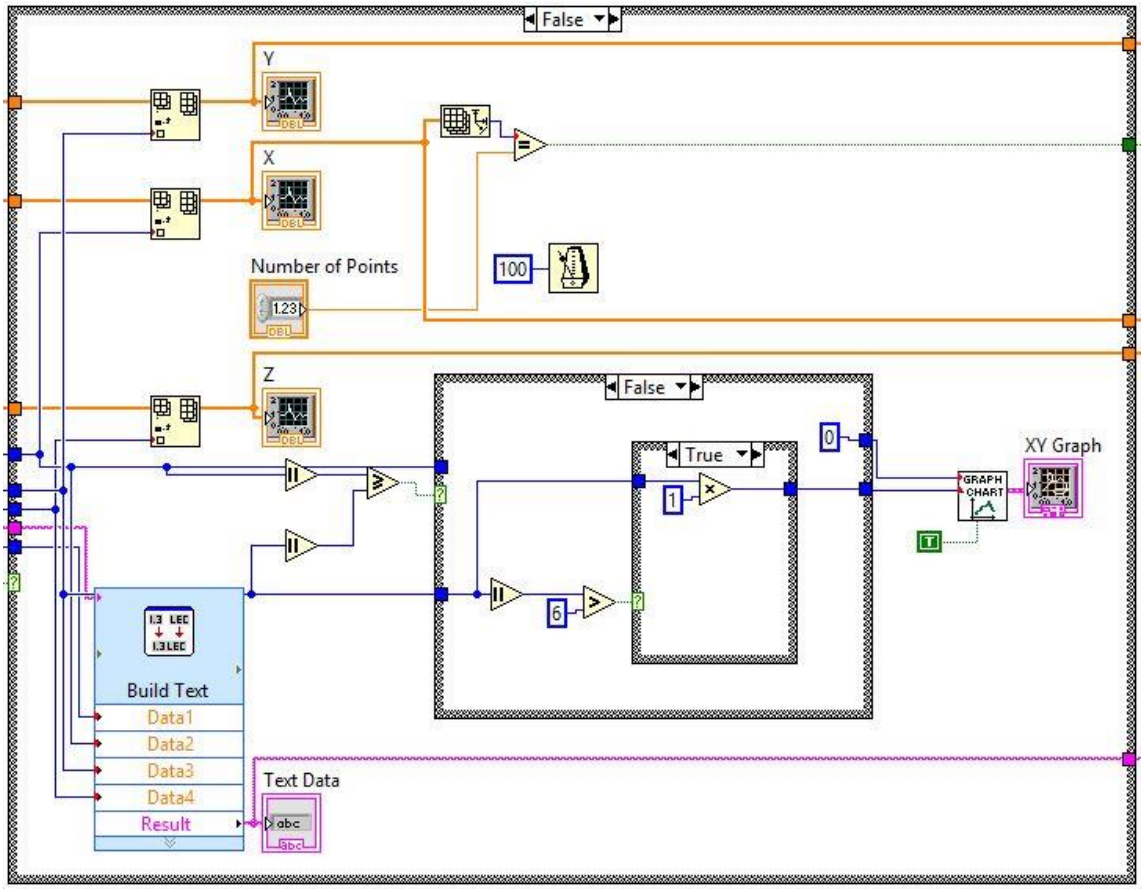

Fig. II.a. Algoritmos de preprocesado y filtrado en LabVIEW (1)

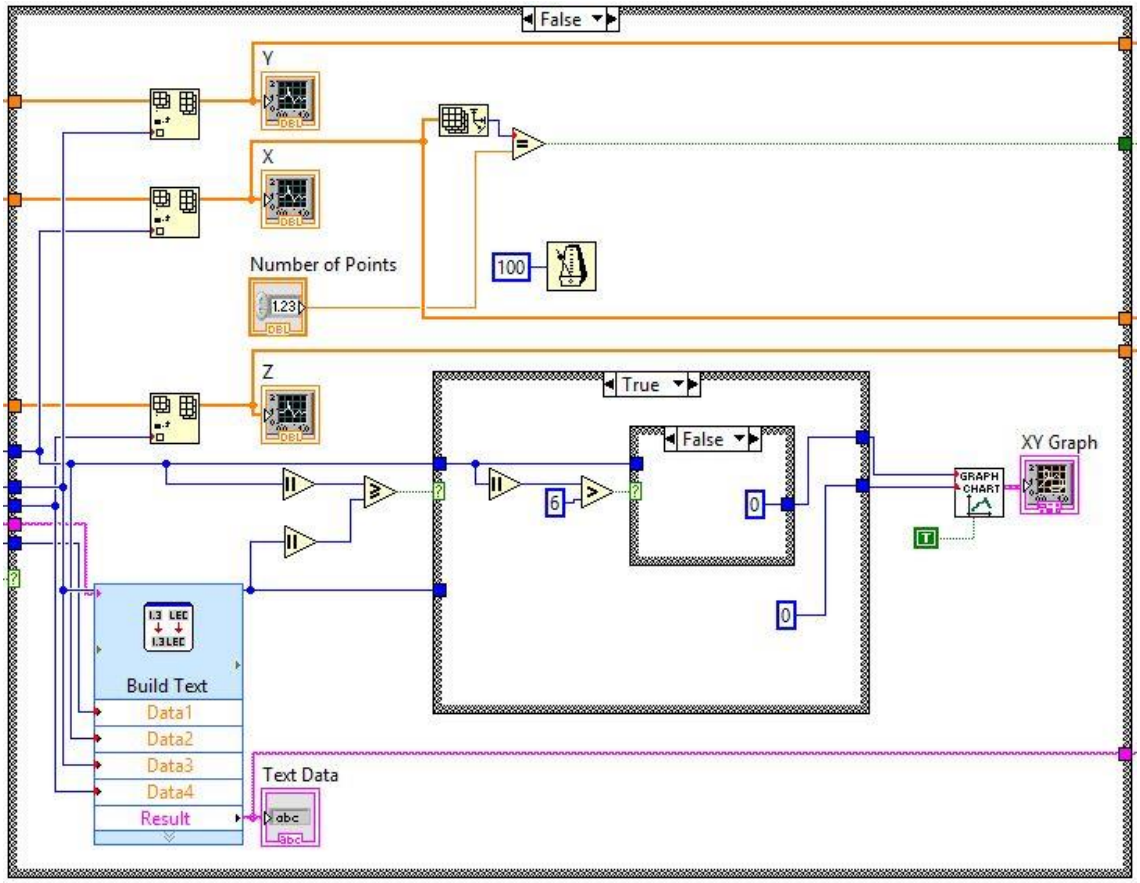

Fig. II.b. Algoritmos de preprocesado y filtrado en LabVIEW (2)

ANEXO III. CRAZYFLIE 2.0

En este anexo se incluyen las características técnicas del Crazyflie 2.0 junto con un programa en Python que para controlarlo mediante la API *cflib*. Además, se explican las funciones que realiza el programa, así como los comandos que se envían al dron.

Características técnicas

Especificaciones mecánicas:
- Peso: 27 g
- Medidas: 92x92x29mm (motor a motor incluyendo los soportes)

Especificaciones de la radio:
- 20 dBm, radio con rango probado de > 1 km.
- Radio compatible con las versiones previas de Crazyflie y Crazyradio

Microcontroladores:
- STM32F405 controlador principal MCU (Cortex-M4, 168MHz, 192kb SRAM, 1Mb flash)
- nRF51822 radio y control de suministro de energía MCU (Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash)

Conector USB:
- Cargador de batería LiPo.
- Puerto USB de alta velocidad.

IMU:
- Giróscopo de 3 ejes (MPU-9250)
- Acelerómetro de 3 ejes (MPU-9250)
- Magnetómetro de 3 ejes (MPU-9250)
- Sensor de presión de alta precisión (LPS25H)

Especificaciones de vuelo:
- Tiempo de vuelo: 7 minutos
- Tiempo de carga de la batería: 40 minutos
- Max recommended payload weight: 15 g

Conector de expansión con:
- VCC (3.0V, máx. 100mA)
- GND
- VCOM (unregulated VBAT o VUSB, máx. 1A)
- VUSB (para input y output)
- I2C (400kHz)
- SPI
- 2 x UART

- 4 x GPIO/CS para SPI
- 2 x GPIO conectado a nRF51

8KB EEPROM

Programa en Python para controlar el Crazyflie 2.0

En este apartado se incluye un pequeño código en Python para controlar el Crazyflie 2.0. Las dependencias necesarias son las librerías *time*, *sys*, *threading*, *logging* y el cliente *cflib* elaborado por Bitcraze.

El programa se conecta con el primer Crazyflie 2.0 que encuentre, y seguidamente activa el modo de altitud constante, es decir envía el comando de hover durante el número de segundos que marque la variable `interval` que se encuentra en el método privado `_hover(self)`. Finalmente, una vez pasado ese tiempo se cierran las vías de comunicación.

```
import time
import sys
from threading import Thread
import logging


sys.path.append("../src/cflib")
import cflib.crtp
from cflib.crazyflie import Crazyflie  # noqa

logging.basicConfig(level=logging.ERROR)


class HoverTest:
"""Example that connects to a Crazyflie and sets the hover mode during 30
seconds and then disconnects"""

    def __init__(self, link_uri):
        """ Initialize and run the example with the specified link_uri """

        self._cf = Crazyflie()

        self._cf.connected.add_callback(self._connected)
        self._cf.disconnected.add_callback(self._disconnected)
        self._cf.connection_failed.add_callback(self._connection_failed)
        self._cf.connection_lost.add_callback(self._connection_lost)

        self._cf.open_link(link_uri)

        print("Connecting to %s" % link_uri)

    def _connected(self, link_uri):
        """ This callback is called form the Crazyflie API when a Crazyflie
        has been connected and the TOCs have been downloaded."""

        # Start a separate thread to do the motor test.
        # Do not hijack the calling thread!
        Thread(target=self._hover).start()

    def _connection_failed(self, link_uri, msg):
```

```
        """Callback when connection initial connection fails (i.e no Crazyflie
        at the specified address)"""
        print("Connection to %s failed: %s" % (link_uri, msg))
    def _connection_lost(self, link_uri, msg):
        """Callback when disconnected after a connection has been made (i.e
        Crazyflie moves out of range)"""
        print("Connection to %s lost: %s" % (link_uri, msg))

    def _disconnected(self, link_uri):
        """Callback when the Crazyflie is disconnected (called in all cases)"""
        print("Disconnected from %s" % link_uri)

    def _hover(self):
        thrust = 32767
        pitch = 0.2
        roll = -2
        yawrate = 0

        # Unlock startup thrust protection
        self._cf.commander.send_setpoint(0, 0, 0, 0)
        interval = 120
        time_initial = time.time()
        while (time.time()-time_initial) <= interval:
            self._cf.commander.send_setpoint(roll, pitch, yawrate, thrust)
            self._cf.param.set_value("flightmode.althold","True")
            time.sleep(0.1)

        self._cf.commander.send_setpoint(0, 0, 0, 0)
        # Make sure that the last packet leaves before the link is closed
        # since the message queue is not flushed before closing
        time.sleep(0.1)
        self._cf.close_link()


if __name__ == '__main__':
    # Initialize the low-level drivers (don't list the debug drivers)
    cflib.crtp.init_drivers(enable_debug_driver=False)
    # Scan for Crazyflies and use the first one found
    print("Scanning interfaces for Crazyflies...")
    available = cflib.crtp.scan_interfaces()
    print(available)
    print("Crazyflies found:")
    for i in available:
        print(i[0])

    if len(available) > 0:
        le = HoverTest(available[0][0])
    else:
        print("No Crazyflies found, cannot run test")
```

ANEXO IV. AR.DRONE 2.0

En este anexo se adjuntan las características técnicas del AR.Drone 2.0, además de un programa en Python que muestra cómo controlarlo mediante la biblioteca *pyardone*. Junto al programa se incluye una explicación de las funciones que se realizan, así como los comandos enviados al dron.

Características técnicas

Asistencia electrónica:

- Procesador: Procesador 1 GHz 32 bits ARM Cortex A8 con DSP vídeo 800 MHz TMS320DMC64x
- Sistema operativo: Linux 2.6.32
- RAM: DDR2 1 GB a 200 MHz
- USB: USB 2.0 de alta velocidad para las extensiones
- Wi-Fi: Wi-Fi b g n
- Giróscopo: 3 ejes, precisión de 2000°/segundo
- Acelerómetro: 3 ejes, precisión de ± 50 mg
- Magnetómetro: 3 ejes, precisión de 6°
- Sensor de presión: Precisión de ± 10 Pa
- Sensores de ultrasonidos para medir la altitud: Medición de la altitud
- Cámara vertical: QVGA 60 FPS para medir la velocidad en vuelo

Motores:

- 4 motores sin escobillas de tipo "inrunner": 14,5 vatios y 28 500 rpm
- Rodamiento de bolas en miniatura: Sí
- Engranajes Nylatron: Sí
- Rodamiento de bolas autolubricante de bronce: Sí

Grabación de vídeo HD:

- Cámara HD: Cámara HD 720p 30 FPS
- Objetivo: Objetivo gran angular: diagonal 92°
- Perfil de codificación básica: H264
- Formato fotos: JPEG
- Conexión: Wi-Fi

Peso:

- Con carena interior: 380 g

Programa en Python para controlar el AR.Drone 2.0

En este apartado se incluye un pequeño código en Python para controlar el AR.Drone 2.0. Las dependencias necesarias son las librerías *time*, *sys* y *pyardrone,* en esta última se encuentran todas los métodos que permiten interactuar con el dron.

En primer lugar el programa se encarga de compensar el AR.Drone 2.0, y seguidamente lee los datos de navegación. Cuando comprueba que el dron está en el suelo, envía el comando de despegue. Una vez se ha realizado el despegue, el programa activa el modo de altitud constante durante 10 segundos.

Pasado ese tiempo, envía un comando al dron para que se mueva hacia la izquierda con una velocidad del 10% durante 2 segundos, lo cual corresponde a 1 metro de desplazamiento. Después se envía un comando con los mismos parámetros de velocidad y tiempo, pero esta vez para que el dron se mueva hacia la derecha, de forma que vuelve a la posición inicial.

Por último el código vuelve a activar el modo de altitud constante durante 5 segundos, y luego hace aterrizar el cuadricóptero.

```
import time
import sys
from pyardrone import ARDrone

drone = ARDrone()

drone.trim()
drone.navdata_ready.wait()  # wait until NavData is ready

while not drone.state.fly_mask:
    drone.takeoff()
drone.hover()
time.sleep(10)

initial_time=time.time()
while time.time()-initial_time<=2:
    drone.move(left=0.1)

initial_time = time.time()
while time.time()-initial_time<=2:
    drone.move(right=0.1)
drone.hover()
time.sleep(5)

while drone.state.fly_mask:
    drone.land()

if drone.state.fly_mask == False:
    sys.exit(0)
```

ANEXO V. IMPLEMENTACIÓN DE *MACHINE LEARNING*

En este último anexo, se incluye un programa en Python donde se implementa *machine learning*.

La función principal del programa, es representar gráficamente los datos de aceleración del reloj eZ430-Chronos en los ejes X e Y, además de detectar cuando se efectúa una palmada.

Primeramente, se carga la base de datos con las palmadas que se usará para el aprendizaje, y a continuación se define el árbol de decisión como algoritmo clasificador. Después, se establecen las comunicaciones y se envían paquetes de petición de datos al reloj.

Cuando se reciben los datos de aceleración, se pasan por el algoritmo clasificador para ver si se ha hecho alguna palmada. Una vez ejecutado el algoritmo de detección, se muestra un gráfico que representa las aceleraciones en los ejes X e Y. Si se ha detectado alguna palmada el fondo del gráfico cambia de color a azul.

```
import matplotlib.pyplot as plt
import time
import numpy as np
from libs import communications, filterings, graphics, datalog
from sklearn.preprocessing import normalize
from sklearn.tree import DecisionTreeClassifier

communication = communications.CommunicationManager()
filtering = filterings.FilteringManager()
graphic = graphics.GraphicsManager()
report = datalog.DatalogManager()

# Load dataset
train_digits = np.genfromtxt('data/clapping.csv', delimiter = ';')
train_data = train_digits[:, :-1]
train_labels = train_digits[:, -1]

train_data=normalize(train_data)

# Create a classifier
classifier = DecisionTreeClassifier(random_state=0)

# Train the classifier
classifier.fit(train_data, train_labels)


class FreeMovement():
    communication.open_serial_port()

    max_samples = 10
    watch_samples_counter = -1

    lower_index = 0
    higher_index = 30
```

```
old_prediction=0

acceleration=[]
x_axis_acceleration = []
y_axis_acceleration = []
z_axis_acceleration = []
test_digits = []
time_limit = 20 # Datalog time
report.create_file('probabilities.txt')

time_initial = time.time()
graphic.set_plot_parameters()

while time.time() - time_initial <= time_limit:
    time_final = time.time()
    bytes_to_read = communication.send_data_request()
    inbyte = communication.read_data(bytes_to_read)
    enter = filtering.data_availability(bytes_to_read, inbyte)

    if (bytes_to_read == 7 and inbyte[3] == 1) or (bytes_to_read
     == 14 and inbyte[10] == 1):
        watch_samples_counter += 1

        x_axis_acceleration.append(inbyte[bytes_to_read-3])
        y_axis_acceleration.append(inbyte[bytes_to_read-2])
        z_axis_acceleration.append(inbyte[bytes_to_read-1])

        filtering.filter_acceleration(x_axis_acceleration,
                                      watch_samples_counter)
        filtering.filter_acceleration(y_axis_acceleration,
                                      watch_samples_counter)
        filtering.filter_acceleration(z_axis_acceleration,
                                      watch_samples_counter)

        acceleration=[x_axis_acceleration[watch_samples_counter],
                      y_axis_acceleration[watch_samples_counter],
                      z_axis_acceleration[watch_samples_counter]]

        test_digits = test_digits +
                   x_axis_acceleration[watch_samples_counter],
                   y_axis_acceleration[watch_samples_counter],
                   z_axis_acceleration[watch_samples_counter]]

        if watch_samples_counter >= 9 and higher_index <=
         len(test_digits):
            # Load the dataset
            test_data =
             normalize(test digits[lower index:higher index])

            # Predict values
            test_predicted = classifier.predict(test_data)
            test_probabilities =
             classifier.predict_proba(test_data)

            if test_predicted == 1 and old_prediction==0:
                graphic.change_color()
                lower_index += 20
                higher_index += 20
            else:
                graphic.restore_color()
```

```
                    lower_index += 3
                    higher_index += 3

                report.record_data('probabilities.txt',
                 test_predicted,
                 x_axis_acceleration[watch_samples_counter],
                 y_axis_acceleration[watch_samples_counter],
                 z_axis_acceleration[watch_samples_counter])

                old_prediction=test_predicted

            graphic.plot_data(
                x_axis_acceleration[watch_samples_counter],
                y_axis_acceleration[watch_samples_counter])

        plt.pause(0.05)  # 50 milliseconds

    graphic.plot_close()
    communication.close_serial_port()
```